DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

An investigation of AI methods for the facilitation of injury diagnosis and treatments

Eine Untersuchung von KI-Methoden zur Erleichterung der Diagnose und Behandlung von Verletzungen

Authors:	Aleksandra Topalova and Julian Kraus
Supervisor:	Prof. Dr. Pramod Bhatotia
Advisor:	Prof. Dr. Bernd Brügge
Submission Date:	15.08.2024

This bachelor's thesis in informatics is a collaborative effort between Julian Kraus and Aleksandra Topalova. The respective contributions are marked with the abbreviations 'JK' and 'AT' at the section titles. The abbreviations are not repeated in subsections if they are already indicated in the main section title.

Acknowledgments

We would like to express our deepest appreciation to our thesis advisor Prof. Dr. Bernd Brügge for his irreplaceable mentorship. The success of this thesis we owe to his dedication and insightful feedback. We would also like to thank Prof. Dr. Pramod Bhatotia for the opportunity to write our thesis at his chair.

We are deeply grateful to Prof. Dr. Jürgen Höher and the entire staff of Sportsclinic Cologne. Their generosity in allowing us to collaborate with them on this topic and their support in both technical and organizational matters were invaluable. Our interactions with Prof. Höher and the clinic provided crucial insights that greatly contributed to the development of our thesis.

Lastly, we would like to acknowledge Prof. Dr. Werner Plötz for the experience and knowledge we gained through our joint efforts on previous projects. Collaborating with Prof. Plötz provided us with valuable experience in the domain of our thesis, which intersects medicine and informatics.

Abstract

In orthopedic care, timely diagnosis and treatment of injuries like meniscus and cruciate ligament tears are critical to preventing further complications. However, patients often face delays of 4–12 weeks before they can be seen by a specialist, which can exacerbate untreated injuries. For instance, unaddressed meniscal root tears can significantly impair knee functionality, stability, and load balancing, while delays in treating anterior cruciate ligament (ACL) tears can increase the risk of subsequent meniscal tears by 2% for every week of delay preoperatively.

This thesis investigates the use of two AI algorithms, Long Short-Term Memory (LSTM) networks and Large Language Models (LLMs), to facilitate the diagnosis and prioritization of treatment for meniscus and cruciate ligament tears. These AI approaches aim to assess the severity and risk associated with specific diagnoses differentiating between high and low risk. The methodology involves mapping a medical dataset to language that could be generated by patients, enhancing the AI's ability to work with patient questionnaire responses and evaluating the effectiveness of the algorithms in classifying the risk levels of injuries based on medical data.

Our contribution lies in the comparative analysis of the AI strategies for classifying patient risk and in the development of a novel approach to translating medical data into patient-generated descriptions. Our best risk predictor using the LSTM algorithm achieved an accuracy of 78% and macro F1 score of 74%.

Contents

1 Introduction

The applicability of Artificial Intelligence (AI) to medical diagnosis has been explored by scientists since the first developments in the field. Currently many medical facilities aim to equalize their prioritization policies and better manage their capacities and demand by using AI's capabilities to give preliminary diagnoses or predict patient risk levels. (Sæther et al., 2019) (O'Beirne et al., 1984) (Johannessen & Alexandersen, 2018)

1.1 History

1.1.1 Artificial Intelligence - JK

Artificial Intelligence is the ability of machines to simulate the human learning process and intelligence (McCarthy et al., 2006). Natural Language Processing is the area of artificial intelligence which explores how machines can understand and communicate using human language. Chomsky's Generative Grammar laid the foundation for its formalization (Chomsky, 1957). The first real advancements of AI in medicine started in the early 1970s in the area of Natural Language Processing with the invention of expert systems. These rule-based systems were based on a knowledge base provided by experts in the field (Lucas & Gaag, 1991). One of the first medical expert systems was MYCIN, which was used to identify bacteria causing infections and recommend the necessary antibiotics (Shortliffe, 1977). Even though MYCIN could explain its result and achieve good results, it was not used in practice, because of the issue of responsibility in case of mistakes. In 1976 CASNET, a new type of expert system, was developed. It uses a causal-associational network, which can represent more complicated relations between symptoms and in turn possesses a more accurate knowledge base (Kulikowski et al., 1982). The expert system DXplain produces possible diagnoses when given a list of symptoms. DXplain uses a knowledge base with initially 500 diseases and has since been extended to more than 2200 (Elkin et al., 2010).

1.1.2 Machine Learning - AT

Machine learning models identify patterns and make decisions based on algorithms and statistical computation with minimal human intervention. It became the foundation of IBM's DeepQA, a software that uses natural language processing (NLP) to communicate in a question-answer manner. One example is Watson, which identified additional RNA-binding proteins altered in amyotrophic lateral sclerosis (Ni et al., 2017). A generic pediatric consultant Pharmabot based on Watson was presented in 2014 (Ni et al., 2017). Later in 2017, Mandy, a patient intake and management chatbot, was introduced (Ni et al., 2017).

1.1.3 Deep Learning - AT

Initial neural networks used only one layer, a collection of neurons, which processes input data. Deep Learning (DL) models use multiple layers, allowing them to learn complex relationships from input data. Some of the most utilized DL methods are Recurrent Neural Network (RNN), Long Short-Term Memory Networks (LSTMs) and Convolutional Neural Networks (CNN). The main feature of RNNs are additional connections between nodes, which form directed cycles and allow the model to retain recent information. RNNs commonly used for sequential data like text, but have problems with long-term dependencies between important information in the text. (Rumelhart et al., 1988) LSTMs are a subtype of RNNs, addressing the problem of capturing long-term dependencies. They have a dedicated cell memory and regulate which information is stored by using input, forget, and output gates. (Hochreiter & Schmidhuber, 1997) LSTMs are highly useful to work with complex data, such as time series, text and speech. CNNs are mainly used for image tasks, since they are made to work on grid-like data. They use convolutional layers to capture the information, the results are then forwarded to pooling layers which are used to downsample the information. Normally, the last layer is a fully connected layer that aggregates information from all preceding layers to produce the final prediction or classification.(Waibel et al., 1989) The first investigations of Deep Learning in the medical field weren't successful, because it requires computing power, which wasn't available yet. (Kaul et al., 2020) Since the 2000s, this limitation has been overcome by the advancement of technology in terms of computing power. In medicine, Convolutional Neural Networks are mostly used to classify images, to detect tumors and other diseases. CNNs have been used to identify melanoma skin cancer, achieving similar results to trained experts. (Esteva et al., 2017) Deep Learning has not only been used for imaging, there have also been successful approaches using NLP models, such as Long Short-Term Memory (LSTM) to perform multi-label disease diagnosis. (Lipton et al., 2015)

1.1.4 Generative Artificial Intelligence and Large Language Models - AT

Discriminative models make predictions based on their input data. Generative Artificial Intelligence (GenAI) on the other hand uses generative models, which produce new information based on data that they have been trained on. (I. J. Goodfellow et al., 2014) Transformers are neural networks with more than two layers which are specialized for understanding the relations between sequential data, like text. They use a mathematical mechanism, attention or self-attention, to compute how important the words in a sentence are. (Vaswani et al., 2017) Transformers-based models, which are pre-trained on vast amount of data and specialize in natural language processing tasks, are called Large Language Models (LLMs). (Openai et al., 2018) Eminent LLM examples are the BERT and GPT model families. The GPT family consists of transformers predicting and generating text, while the BERT family is focused on understanding the context of words and making predictions for various NLP tasks. In the medical field, BERT has been used to predict clinical diagnosis from Patients Electronic Health Records (Blinov et al., 2020) and the chatbot version of GPT, ChatGPT, was also explored as a diagnostic tool (Kuroiwa et al., 2023). Another LLM example is Med-PaLM, a model

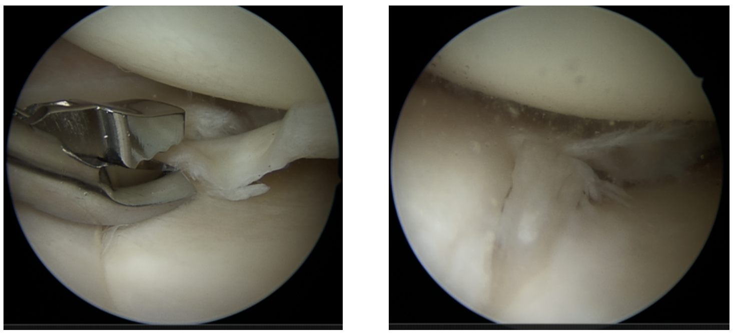


Figure 1.1: Example pictures of an Arthroscopy of a left knee

developed by Google that has multiple medical applications. Because it was trained on a large corpus of medical question-answer data, it can answer both medical exam and patient questions. This makes it capable of performing disease diagnosis and serve as a healthcare chatbot. (Singhal et al., 2023)

1.2 The Problem of untreated injuries - AT

The focus of this thesis is on meniscus and cruciate ligament tears. Patients often wait 4–12 weeks for their appointment due to the limited number of appointments available. During the appointment, the doctor diagnoses the patients and recommends them a treatment, which is normally either operative (surgical) or conservative (non-surgical). In orthopedics, such appointment delays cause untreated or untimely diagnosed injuries, which in turn could lead to health complications. For example, unaddressed meniscal root tears drastically impair the knee functionality, its stability, and load balancing; anterior cruciate ligament (ACL) tears increase the risk of meniscal tear by 2% for every week of delay preoperatively(Strauss et al., 2016). Making it important to correctly prioritize patient appointments by estimating how high-risk their injuries are. Determining the risk level, however, is a complex task that involves taking into account the size, location, and pattern of the tear which dictates the treatment options. For example, intrasubstance/incomplete meniscus tears occur over time, because they are caused by wear and tear, and generally do not require a surgery (low-risk injuries). On the other hand, bucket-handle tears commonly occur under traumatic circumstances and have to be surgically treated (high-risk injuries).

The hypothesis in this thesis is that Artificial Intelligence methods can predict the risk level of orthopedic injuries and be used to prioritize care of patients with critical symptomatic. In this thesis, we focus on meniscus and cruciate ligament tears. The results would be based on patient description of their current condition. We compare two algorithms: Long Short-Term Memory (LSTM) and Large Language Models (LLMs). This thesis follows the OOSE (Object-Oriented Software Engineering) methodologies as described by Bruegge and Dutoit, 2010. Moreover, we are collaborating with Prof. Dr. med. Jürgen Höher and the staff of Sportsclinic Cologne, who provided us with medical patient data for the purpose of risk prediction.

2 Terminology

2.1 Deep Learning - JK

Activation Function. *Activation functions* are used to bring non-linearity to each neuron in a neural network (Apicella et al., 2021) (Ding et al., 2018) (Sharma et al., 2017). The output of a basic neuron is defined as follows:

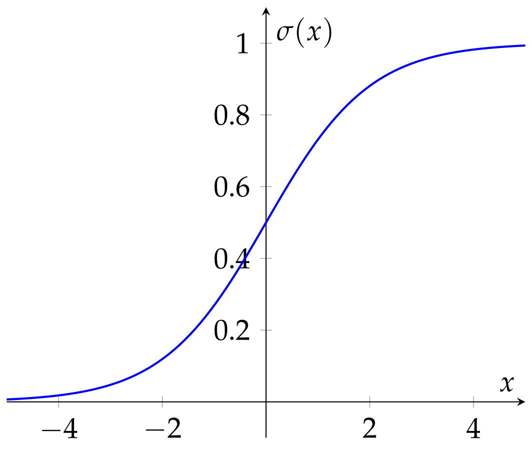
$$output = act(neuronState) = act(weight * input + bias)$$

Sigma function. The *Sigma function* maps an input to a value between 0 and 1. It is a smooth differentiable function, and often used for gradient based optimization algorithms. (Apicella et al., 2021) (Ding et al., 2018) (Sharma et al., 2017)

Mathematical Expression of the Sigma Function:

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Graphical Representation of the Sigma Function: The graph below illustrates the behavior of the Sigma function, mapping input x ranging from -5 to 5 to an output between 0 and 1:

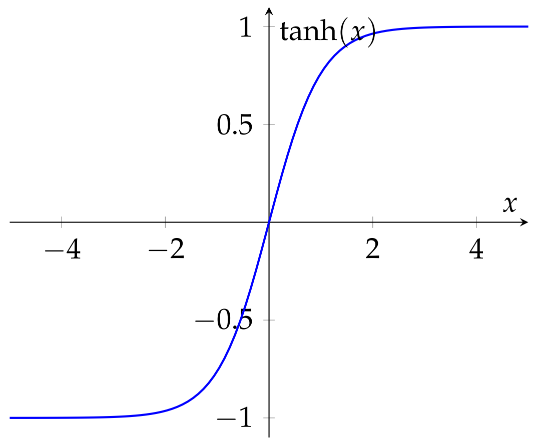


Hyperbolic tangent. The *tanh* is a smooth differentiable activation function, which maps a number to a value between -1 and 1. It is used to center the input data and scale it. (Apicella et al., 2021) (Ding et al., 2018) (Sharma et al., 2017)

Mathematical Expression of the tanh Function:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

Graphical Representation of the tanh Function: The graph below illustrates the behavior of the tanh function, mapping input x ranging from -5 to 5 to an output between -1 and 1:

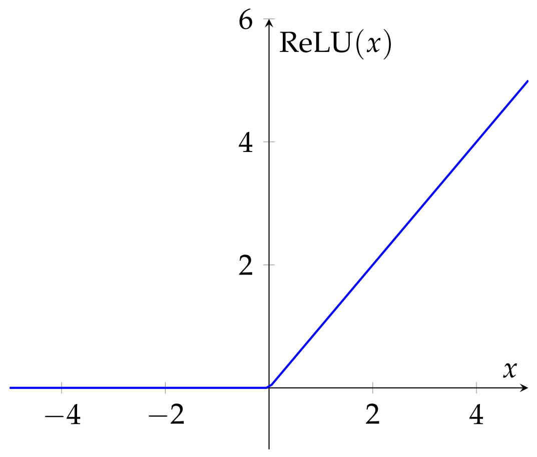


ReLU function. The *ReLU function* is a non-linear activation function, which maps the input to value between 0 and infinity. It can mitigate vanishing gradient because it is not restricted to one side. (Apicella et al., 2021) (Ding et al., 2018) (Sharma et al., 2017)

Mathematical Expression of the ReLU Function:

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

Graphical Representation of the ReLU Function: The graph below illustrates the behavior of the ReLU function, mapping input x ranging from -5 to 5 to an output between 0 and positive infinity:

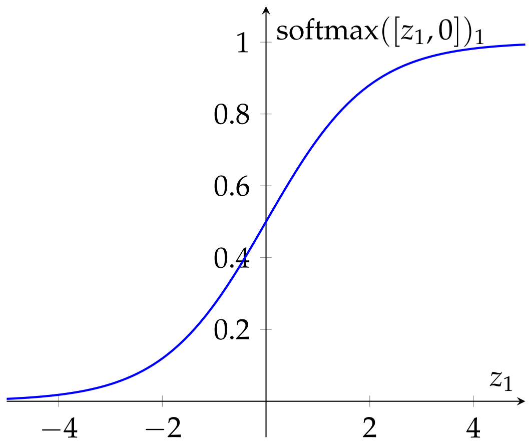


Softmax function. The *softmax function* is a combination of multiple sigmoid functions. Every sigmoid function returns a value between 0 and 1. For classification tasks, this result can be used as the probability for the specific classification. (Ding et al., 2018) (Sharma et al., 2017)

Mathematical Expression of the Softmax Function:

$$\text{softmax}(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} for j = 1, ..., n$$

Graphical Representation of the Softmax Function: The graph below illustrates the behavior of the Softmax function, mapping input x ranging from -5 to 5 to an output between 0 and 1:



2.2 Evaluation

Loss - JK. *Loss* signifies the difference between the prediction and the actual result, which makes 0 a the optimum. Loss is a useful metric for evaluating the performance during model training. (Boer et al., 2005)

Cross Entropy Loss - AT. *Cross entropy loss* (CE Loss) measures the difference between a predicted and an actual probability. LLMs often use mean aggregated cross entropy loss to optimize their performance by calculating the average loss across a sequence of words or sub-words that they predicted. (Boer et al., 2005) (Grandini et al., 2020)

$$\text{Mean aggregated CE} = -\frac{1}{S}\sum_{i} \log(p_{t_i}),$$

- S is the sequnce length
- i is the position in the sequence
- t_i is the correct label for a word or a sub-word at a position i
- p_{t_i} is the predicted probability for the correct word or sub-word

Categorical Cross Entropy Loss - JK. *Categorical cross entropy loss* is commonly used for multi-class classification tasks. It is based on cross entropy, but additionally the actual value for the classes is encoded as a one-hot vector. The loss for all data points is then calculated as the average over the singular points. (W. Liu et al., 2016)

Mathematical Expression of the Categorical Cross Entropy Loss:

$$L = \sum_{j=1}^{M} l_j \text{ with } l_j = -\sum_{i=1}^{N} y_{ij} \log(p_{ij})$$

- N is the number of classes, M number of data points
- l_j is the loss of element j
- L total loss
- y_{ij} is the correct label for class i [0, 1]
- p_{ij} is the predicted probability for class i [0–1]

Confusion Matrix - AT. A *confusion matrix* visualizes classification results from classification in a square heatmap, where the x- and y- axis are both labelled with the same classification labels. The rows represent the actual instances of a given class in the original dataset and the columns contain the predicted instances. The diagonal in the matrix contains the accurate predictions. (Grandini et al., 2020) (Vujović, 2021) (M & M.N, 2015)

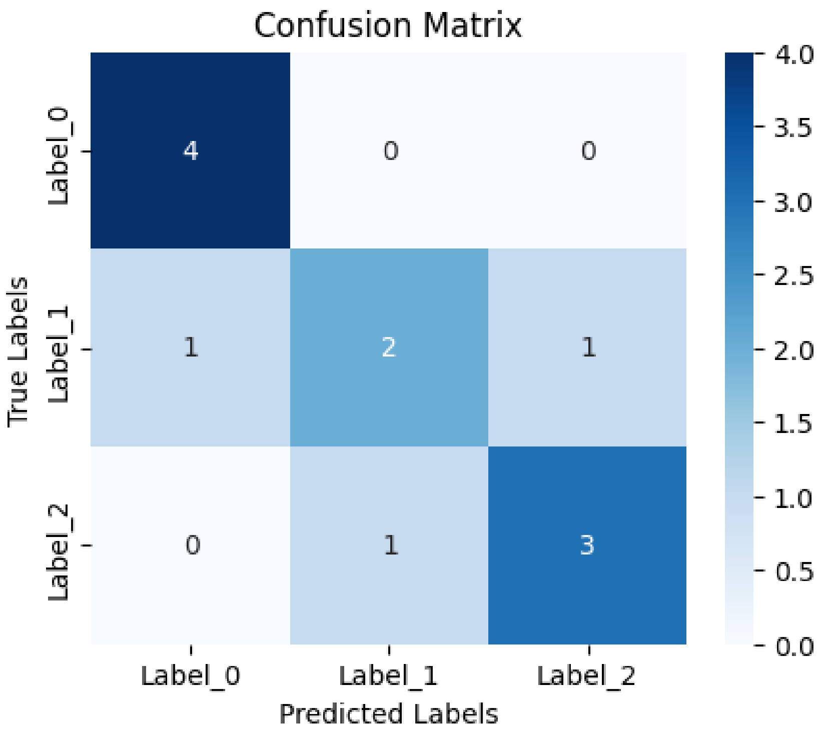


Figure 2.1: Example confusion matrix of a classification with Label 0, Label 1 and Label 2

Accuracy - JK. The *accuracy* describes the ratio of correct predictions over all predictions. Accuracy ranges from 0 to 1, with 1 representing the optimal value. (Grandini et al., 2020) (Vujović, 2021) (M & M.N, 2015)

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Error Rate - JK. In contrast to the accuracy, *error rate* measures the ratio of wrong predictions to all predictions. Error rate ranges from 0 to 1, with 0 representing the optimal value. (Grandini et al., 2020) (Vujović, 2021) (M & M.N, 2015)

$$\text{Error Rate} = \frac{FP + FN}{TP + TN + FP + FN}$$

Precision - JK. *Precision* measures the precision of correctly identifying positive predictions, meaning the ratio of correct positive predictions of all positive predictions. Precision ranges from 0 to 1, with 1 representing the optimal value. (Grandini et al., 2020) (Vujović, 2021) (M & M.N, 2015)

$$\text{Precision} = \frac{TP}{TP + FP}$$

Recall - JK. On the other hand, *recall* denotes the fraction of correct positive predictions that are correctly recalled as positive. Recall ranges from 0 to 1, with 1 representing the optimal value. (Grandini et al., 2020) (Vujović, 2021) (M & M.N, 2015)

$$\text{Recall} = \frac{TP}{TP + FN}$$

Specificity - JK. *Specifcity* measures the fraction of correct false values, from the ones predicted as false. Specificity ranges from 0 to 1, with 1 representing the optimal value. (Grandini et al., 2020) (Vujović, 2021) (M & M.N, 2015)

$$\text{Specificity} = \frac{TN}{TN + FP}$$

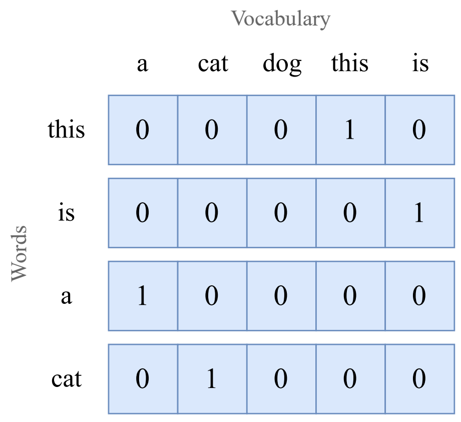
F-Measure - AT. The *F-Measure*, also knows as F1, represents the harmonic mean between precision and recall. It is often used for imbalanced datasets. The FM ranges from 0-1 and is optimal when it reaches 1. There are three subcategories of F1 - macro, micro and weighted. Macro computes the average of the all the F1 scores of the classes and therefore does not prioritize majority classes in imbalanced datasets. Micro F1 computes the proportion of overall correctly classified entries from all entries using precision and recall, which makes it equal to overall accuracy. Weighted F1 adds all F1 class scores together after weighting them with their frequency in the whole dataset, which is called support. (Grandini et al., 2020) (Vujović, 2021) (M & M.N, 2015)

$$F1_{macro} = \frac{F1_1 + F1_2 + ...F1_n}{n}$$

$$F1_{micro} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

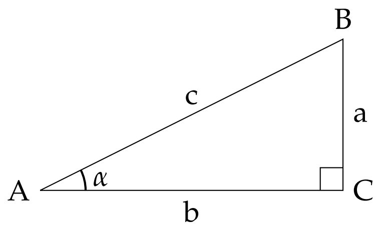
$$F1_{weighted} = F1_1 \cdot S_1 + ... + F1_n \cdot S_n, \text{ where } S_i = \frac{\text{Occurences of class } i}{\text{Total number of entries}}$$

Word vectorization - AT. *Word vectorization* is the process of encoding words or phrases into vectors. There are two common vectorization techniques - one-hot encoding and word embeddings. **One-hot encoding** initially builds a "vocabulary" containing the words from a data corpus of size N, where N is the amount of words. To vectorize a phrase, for every contained word a vector V with length N is built. Internally checks the position of that word in the vocabulary, e.g. i, marks position in the vector with "1" ($V_i = 1$) and fills all remaining indices with "0". These binary vectors are finally connected in a sparse matrix with width N, which does not represent the relationship between words (See Figure 2.2). **Word embedding** techniques, such as Word2Vec (Church, 2017), are trained to create dense word vectors, where each real-number value is calculated to represent a part of the word's context. This multi-dimensional translation of words captures their semantic meaning and relations through the vector distance, angle and direction.


Vocabulary
Words

	a	cat	dog	this	is
this	0	0	0	1	0
is	0	0	0	0	1
a	1	0	0	0	0
cat	0	1	0	0	0

Figure 2.2: Example visualization of one-hot encoding



$$\cos(\alpha) = \frac{b}{c}$$

Figure 2.3: Cosine of angle α in $\triangle$ ABC

Cosine - AT. *Cosine* (*cos*) is a trigonometric function, which for a right triangle $\triangle$ABC and an angle α represents the proportion of the side adjacent to α and the hypotenuse. (Figure 2.3)

Cosine similarity - AT. *Cosine similarity* is a metric, which can be used to determine the similarity between two text structures represented as vectors. The similarity between these two vectors is calculated as the cosine of the angle between them (Figure 2.4), hence the name of the metric. It can be represented with the vectors' dot product and their magnitudes. Therefore, vectors aligned in the exact same direction would have a similarity measurement of 1, because the angle between them is 0° and $cos(0°) = 1$. Similarly, orthogonal vectors have cosine similarity of 0 due to $cos(90°) = 0$. (Han et al., 2012)

$$S_c(A, B) = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|}, \text{ with A and B - non-zero vectors}$$

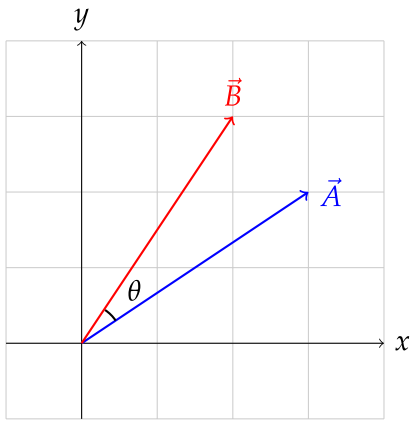


Figure 2.4: Visual representation of Cosine Similarity in a cartesian coordinate system

2.3 Additional Algorithms

K-Means K-means is an unsupervised clustering algorithm that partitions data into k clusters by iteratively assigning points to the nearest centroid and recalculating centroids until convergence. It's efficient for large datasets but sensitive to initial conditions and assumes clusters are spherical and of similar size (Ahmed et al., 2020).

Word2Vec Word2Vec is a neural network-based model that transforms words into continuous vector representations, capturing semantic relationships. The model has two main architectures: Continuous Bag of Words (CBOW) and Skip-gram. Skip-gram predicts surrounding words given a target word, focusing on preserving context by learning word associations. This approach is particularly effective for capturing word meanings and relationships in large text corpora (Mikolov et al., 2013).

3 Requirement Analysis

This chapter models the application domain of the patient prioritization system. Section 3.1 describes the functional model. Section 3.2 goes into detail about the dynamic behavior and section 3.3 describes the analysis object model. These models will be refined in the object design chapter 6.

3.1 Functional Model - AT

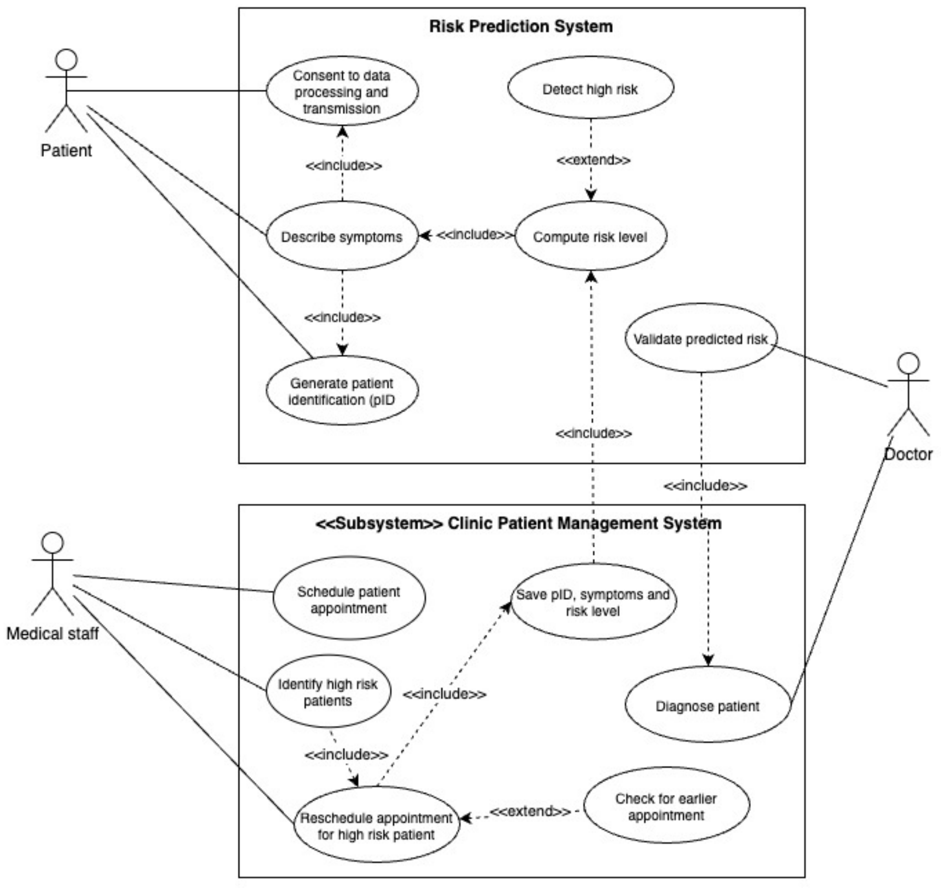

Risk Prediction System
Patient
Consent to data processing and transmission
Detect high risk
<<include>>
<<extend>>
Describe symptoms
<<include>>
Compute risk level
<<include>>
Validate predicted risk
Generate patient identification (pID
<<include>>
Doctor
<<include>>
<<Subsystem>> Clinic Patient Management System
Medical staff
Schedule patient appointment
Save pID, symptoms and risk level
Identify high risk patients
<<include>>
Diagnose patient
<<include>>
Reschedule appointment for high risk patient
<<extend>>
Check for earlier appointment

Figure 3.1: Functional model of the patient prioritization system (UML use case diagram)

The use case diagram in Figure 3.1 shows the association between the actors and the risk diagnosis system. The actor **Patient**, experiences injury symptoms, **Doctor** is an orthopedist who treats the Patient. The Medical Staff actor is responsible for the organizational aspects of the Patient treatment. The **Risk Prediction System**, responsible for predicting the patient risk and the **Clinic Patient Management System** manages the patient details. The Medical Staff schedules an initial patient appointment in the Clinic Patient Management System. The Patient describes their symptoms in the Risk Prediction System, which includes the patient consenting to data processing and transmission and the system generating patient identification (pID). The Risk Prediction System computes the risk level for the patient injury based on the described symptoms and when the injury is severe, it detects high risk. The pID, symptoms and computed risk level are saved in the Clinic Patient Management system. The Medical Staff can identify high-risk patients from the saved data and reschedule the initial appointments for those high-risk patients for an earlier slot if earlier appointments are available. At the appointment, the Doctor diagnoses the Patient in the Clinic Patient Management System and this diagnosis is used to validate the predicted risk by the Risk Prediction System.

3.2 Dynamic behavior - JK

Figure 3.2 describes the dynamic model of the patient prioritization system. The activities of our proposed solution are as follows. First, the **Patient** observes pain in their knee, which leads them to call the clinic. The **Medical Staff** gives the patient an appointment and also mentions that they should fill out the questionnaire in the **Risk Prediction System**. The Risk Prediction System requests the patient identification, consent for data processing and the answers to the questionnaire. It then selects the best risk predictor and computes the risk. The patient info and computed risk are then sent to the Medical Staff. While the Patient is waiting for their appointment, if there is an earlier appointment available, the Medical Staff will reschedule high risk patients to the earlier appointments. When the Patient attends the appointment, they will get diagnosed by the **Doctor**, who then uploads the diagnosis to the Risk Prediction System, to improve the model.

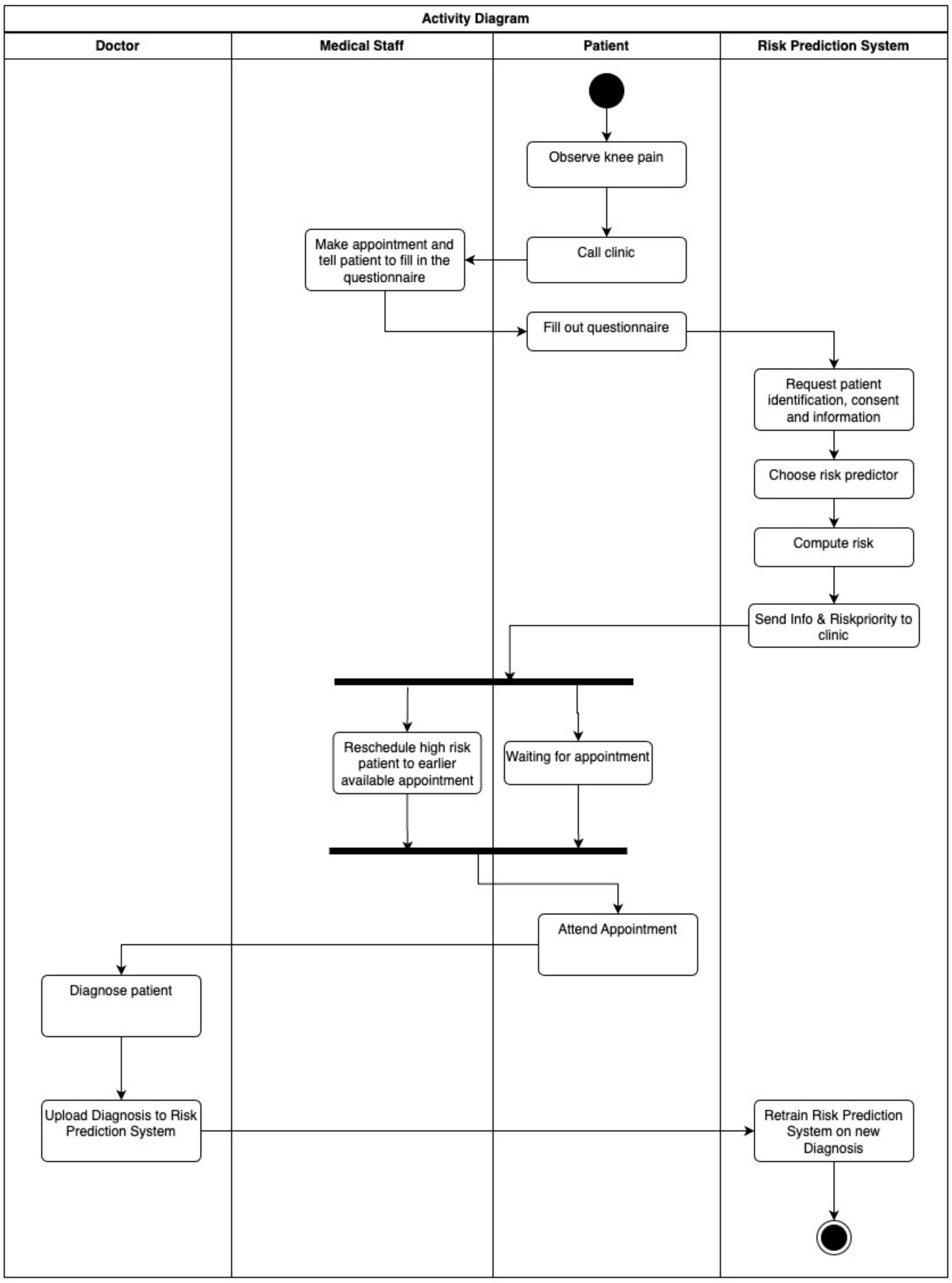

Activity Diagram
Doctor
Medical Staff
Patient
Risk Prediction System
Observe knee pain
Call clinic
Make appointment and tell patient to fill in the questionnaire
Fill out questionnaire
Request patient identification, consent and information
Choose risk predictor
Compute risk
Send Info & Riskpriority to clinic
Reschedule high risk patient to earlier available appointment
Waiting for appointment
Attend Appointment
Diagnose patient
Upload Diagnosis to Risk Prediction System
Retrain Risk Prediction System on new Diagnosis

Figure 3.2: Dynamic behavior of the system using an UML Activity diagram

3.3 Analysis Object Model - AT

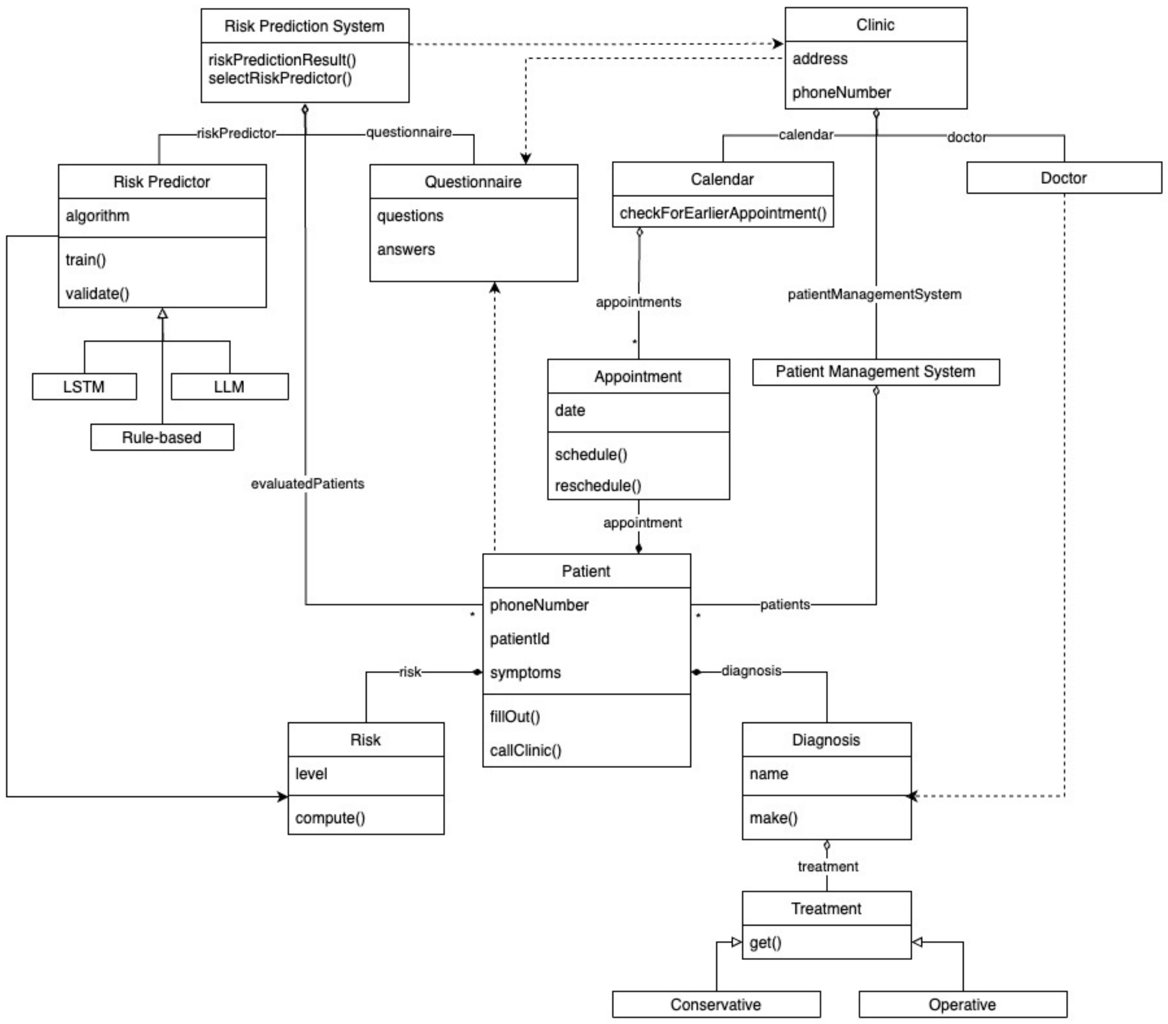


Figure 3.3: Analysis object model (UML class diagram)

Figure 3.3 shows the object model of the patient prioritization system. Each **Patient** is associated with a **Risk** has a level and can be computed. A **Doctor** can make a **Diagnosis**, which has a name and a **Treatment**. A Treatment is either **Conservative**, or **Operative**. An **Appointment** has a date and can be scheduled and rescheduled. The Risk, Appointment and Diagnosis are associated with a Patient, who has a phone number, a patientID and symptoms, and can call the **Clinic** and fill out a **Questionnaire**. The Clinic has an address, a phone number, a **Calendar**, a **Patient Management System** and is associated with one or more Doctors. The Calendar consists of all the appointments and can be used by the Clinic to check for earlier appointments. The Patient Management System comprises all the patients. Finally, the Clinic uses the Questionnaire, which has questions and the answers, filled out by the Patient. The Questionnaire is a part of the **Risk Prediction System** together with the **Risk Predictor**. It can be trained, validated and is responsible for computing the Risk of Patients. The Patients are evaluated by having their risk computed with a Predictor, selected by the Risk Prediction System. The Risk Predictor can be an **LSTM, Rule-based or a LLM**. Risk prediction results can be inquired by the Clinic by invoking the RiskPredictionResult method.

4 Study Design - JK

To achieve the goal of creating an accurate algorithm for risk prediction, example data is necessary. Deep Learning approaches typically require as many examples as possible to achieve optimal results. This chapter describes our approach to the creation of the initial study dataset in section 4.1 and the subsequent filtering, structuring and preparation of the data for use in model training in sections 4.2, 4.3 and 4.4, reducing the data from 5,093,345 entries to 40260 entries. Section 4.5 introduces the questionnaire for the Risk Prediction System Section 4.6 introduces an iPad App, which uses the questionnaire for initial collection of patient responses. Additional approaches for extracting the most important information from the data are described in section 4.7. Any data shown in this chapter has been artificially created to give an impression of the data, while respecting data protection laws.

4.1 Study Dataset creation - JK

The data is from a database of patient data collected in the Sportsclinic Cologne. To retain the privacy of the patients the data is pseudo-anonymized, and preprocessed by filtering the names, address and any other potentially identifying information and only keeping a generic patientID (Tinabo et al., 2009). For the data, we determined the following requirements:

1. Knee-specific, not possibly related to other body parts
2. Collected before an operation occurred
3. No rare injury, meaning more than 1-2 occurrences in a 1.5 year period in the database

Component	Description	Example
Letter	Area of classification	**M** (Diseases of the musculoskeletal system)
Nonspecific	Category/block of diseases	M**23** (Disorders of meniscus)
Specific	Specific condition	M23.**3** (Other meniscus derangements)

Table 4.1: Table detailing the ICD code format

We are collecting the data based on the diagnoses the patients received, which are given as ICD-codes. The codes are structured as described in table 4.1[1] [2]. We differentiate between

[1]https://www.barmer.de/gesundheit-verstehen/medizin/arztbesuch-behandlung/icd-diagnoseschluessel-1070992

[2]https://www.icd-code.de/

specific and nonspecific parts of the codes, focusing only on the nonspecific one for the extraction. Based on the frequency of the codes in the database regarding requirement 3) and relevance according to requirement 1), we determined the following nonspecific ICD codes to focus on.

- M23 - Knee injuries
- S83 - Dislocation, sprain, and strain of joints and ligaments of the knee joint
- M17 - Gonarthrosis [Arthrosis of the knee joint]
- M22 - Diseases of the patella

For each of the selected ICD codes, we extracted information from three different tables found in the clinic database. In total we extracted 5,093,345 entries. In the following, we describe the exported tables and our initial filtering of the data.

4.1.1 General patient information

The general patient information table is structured in entries with several attributes shown in table 4.2.

Attribute	**Description**
Type	Type of the entry
Date	Date of the entry
Text	Content of the entry
Operating Facility	Facility where the entry was entered
Card Type	Type of card used
Media Type	Type of media used
Patient ID	Identification number of the patient
Last Name	Surname of the patient
First Name	First name of the patient
Gender	Gender of the patient
Last User	The last user who modified the entry
Card Entry ID	Unique identifier for the card entry

Table 4.2: Patient Information Attributes

After careful consideration and removing unnecessary attributes, the following essential attributes remain:

- Type
- Date
- Text

- Patient ID
- Gender

The entries are divided into 200 types, including ones relating to medication, administrative information and patient visit records. From those we chose only the anamnesis and examination. The other types were not of interest to us, since they contained information not relevant to the project. From the remaining patient information data we removed any entries containing a reference to an operation, to fulfill requirement 2 defined in section 4.1.

4.1.2 Patient diagnose information

The patient diagnose information table is structured in entries with several attributes shown in table 4.3.

Attribute	**Description**
Code	ICD Code
Abbreviation	Abbreviation of the doctor
ICD Text	Text description from ICD coding
Own Designation	Custom label for internal use
Date	Date of the record entry
DDI	Drug-drug interaction identifier
Patient ID	Identification number of the patient
First Name	First name of the patient
Last Name	Last name of the patient
Birth Date	Date of birth of the patient
Years	Age in years
Gender	Gender of the patient
Document ID	Unique document identifier

Table 4.3: Patient Diagnose Information Attributes

After removing unnecessary attributes, the following key attributes remain:

- Code
- ICD Text
- Own Designation
- Patient ID
- Date
- Gender
- Birth Date

4.1.3 Patient operation information

Initially we also exported and were considering to use the operation information. Upon further investigation we determined that it does not necessarily indicate risk, since the operation might be necessary, but not of high priority. This led us to discarding the information.

4.1.4 Study Dataset structuring - JK

We combined the data of the two tables by merging them together. The birthdate found in the diagnosis table was used to compute the age of the patients at the time of the entry. Leaving us with a table containing the column described in figure 4.4.

Attribute	Description
Code	ICD Code, for any diagnosis entries
Text	Detailed description or notes
Date	Date of the record entry
Patient ID	Identification number of the patient
Years	Age of the patient in years at time of the entry
Gender	Gender of the patient
Entry-Type	Type of card or document category (Diagnose, Anamnesis or Examination)

Table 4.4: Study Dataset Attributes

The combined data we then grouped by patient, creating history for each patient. Since patients might have longer histories with the clinic, we estimated that after visit pauses of at least two years any new visits would relate to a new problem. Below this time any longer pauses between visits could also pertain to post operation checkups. Regarding that, we created patient injury clusters based on all visits clustered together and relating to one injury of the patient. Some of the examinations and anamneses are non-descriptive and would only confuse a model, leading to us removing any such entries under 25 characters. From this data, we removed any clusters not containing a diagnosis, which we need for our risk assessment. Additionally, we removed any clusters where neither an anamnesis nor an examination exists. For our final data we took the first entries for each anamnesis, examination and diagnosis. In the end, the study dataset is comprised by 40260 problem clusters coming from 37617 patients.

4.2 Study Dataset Fields - JK

Our remaining patient data contains several data fields, also described in figure 3.3 in the object model. Figure 4.1 details the structure of the data from this point on. The main focus of this section are the anamneses and the examinations.

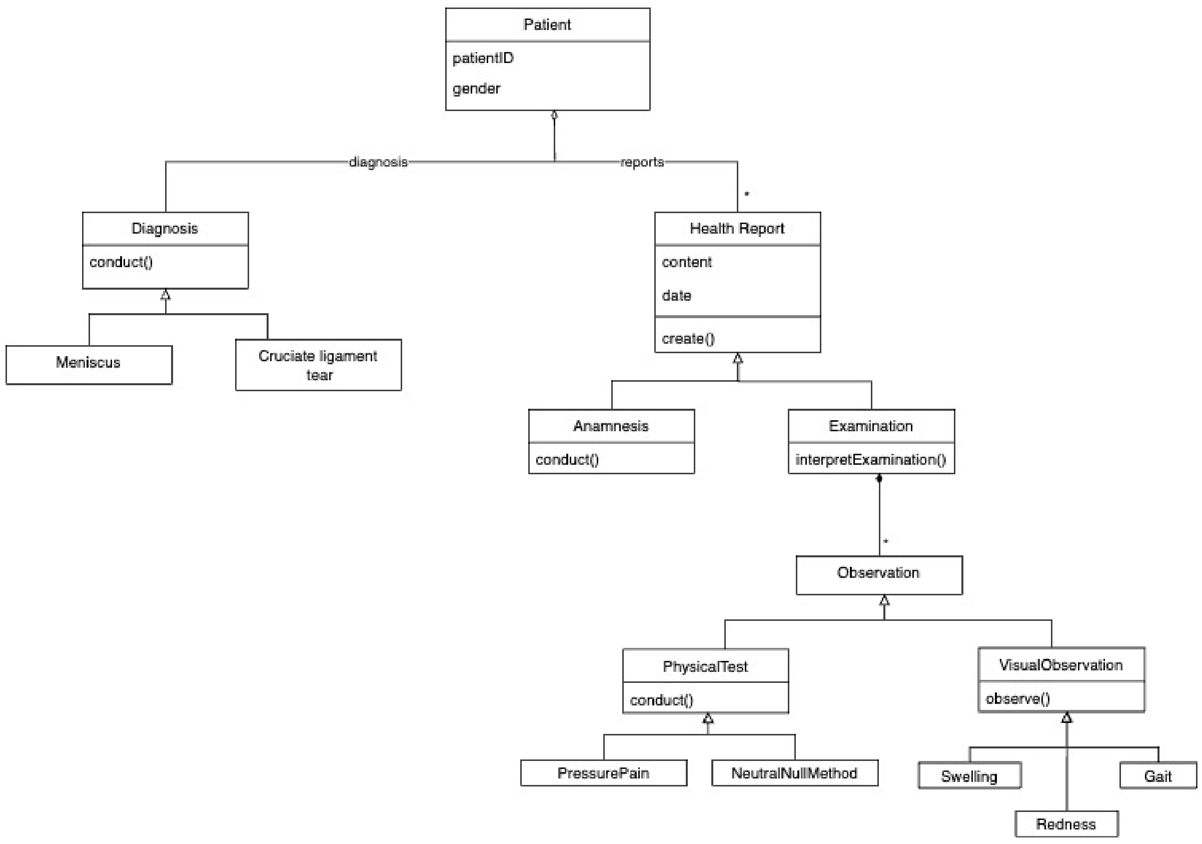


Figure 4.1: Data structure UML class diagram

Anamnesis - JK

In the anamnesis, the doctors describe their patients' symptoms from the patient's point of view and it also includes the patient history relevant to the current symptoms. An anamnesis is written in a semi-factual language, combining medical terminology with predominantly informal words. Two example anamnesis can be seen in figure 4.2.

Examination - JK

An examination described in 4.1 is written in medical jargon, making it unintelligible for people without a medical background. The examination includes visual observations of more complicated symptoms. Additionally, it comprises the results of various orthopedic tests. For example, the notation 5/0/140 describes the joint mobility of the patient at flexion and extension. Two examples can be seen in figure 4.3.

Example 421

Patient reported that he had tried to play basketball again in the meantime and that he would be able to do so again if he warmed up well. But on the following day he still felt a slight discomfort. He still felt a slight blockage and pressure pain.

Example 1

Pain in right knee joint since the beginning of April following internal rotation trauma while doing sports. Treatment to date (rest and orthosis) without relevant improvement. Moderate restriction of mobility, moderate feeling of instability, feeling of twisting. She is currently wearing an orthosis.

Figure 4.2: Patient examples (anamnesis)

Example 421

Leg axis straight Right knee joint: extension/flexion 0/25/110° (springy extension inhibition), no joint effusion, Lachman neg. firm, collateral ligaments stable, pressure pain lateral joint space. inconspicuous patella.

Example 1

Fluent gait pattern under full weight-bearing, left knee: ROM: 0/0/129°, opposite side: 6/0/132°, discrete effusion, Lachmann and pivot shift, instrumental stability measurement (Rolimeter at 25° flexion): uninjured: 7 mm, injured 7 mm, lateral difference: 0 mm, quarter squat in single-leg stance: unstable and with medial collapse MCL stable in lateral comparison.

Figure 4.3: Patient examples (examination)

4.3 Examination interpretation - JK

The factual examination shows little similarity to any information a patient would enter. The examination also contains a number of test results, a patient could not perform themselves. It is questionable if any risk predictor trained on such data would be able to generalize to data entered by patients themselves. To solve this issue, we propose the filtering and interpretation of the examinations to achieve a result akin to patient entered data.

4.3.1 Examination Clustering

The examinations consist of free text, which is parsed into individual observations as illustrated in Figure 4.1, utilizing commas, periods, and semicolons as delimiters. This parsing allows for the segmentation of the text into discrete observations. To gain an overview of the different types of observations, clustering based on similarity is performed. K-Means, a widely-used clustering algorithm, is selected for this initial exploratory analysis. However, it quickly became evident that the simple parsing based on commas and periods often fails due to exceptions, such as semantically-connected comma-separated data fragments, abbreviations ending in periods, and conjunctions like "and" that separate data points.

By evaluating the clustering results, it is possible to identify and exclude certain values from the parsing process, improving the accuracy of observation separation. This evaluation suggests the potential for transforming the examination data by selecting relevant clusters and applying rule-based transformations. The silhouette score is employed to determine the optimal number of clusters.

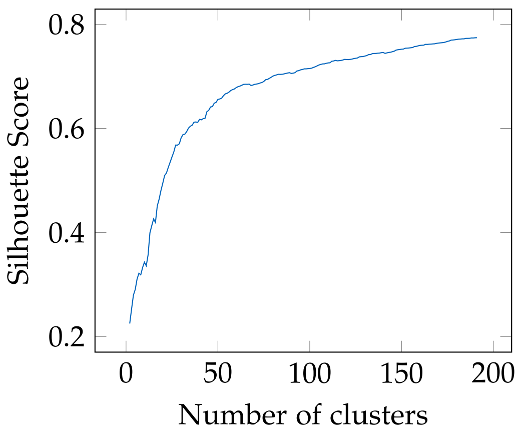


Figure 4.4: Silhouette Score Graph showing an upwards trend even into high number of clusters

The strict upward trend in the silhouette score shown in figure 4.4 is a result of the irregularities and numerous exceptions in the data structure. This trend indicates that clustering alone may not be a suitable method for comprehensive examination interpretation. Relying solely on clustering could lead to either significant information loss within non-specific clusters or necessitate a high number of clusters, making the process overly complex and time-consuming. Despite these limitations, clustering proves valuable as a visualization and structuring tool, particularly in identifying exceptional values that should not be split. An example of examinations split in observations is shown in figure 4.5.

4.3.2 Observation Selection

Based on the clustered observations, we selected several types specified in figure 4.1:

Physical tests:

Example 421

leg axis straight right knee joint extension/flexion 0/25/110° springy extension inhibition, no joint effusion, lachman neg firm, collateral ligaments stable, pressure pain lateral joint space, inconspicuous patella

Example 1

flowing gait pattern under full load, left knee rom 0/0/129°, opposite side 6/0/132, discrete effusion, lachmann, pivot shift, instrumental stability measurement rolimeter at 25 flexion uninjured 7 mm injured 7 mm, lateral difference 0 mm, quarter squat in one-legged stance unstable, with medial collapse mcl stable in lateral comparison

Figure 4.5: Patient examples (examination split in observations)

- The mobility of the knee, specified by the neutral null method (e.g. 0/25/110)
- Pressure pain
- Instability measurements
- Results of the lachman test

Visual observations:

- Swelling or effusion
- Gait of the patient
- Redness
- Impressions by the patient

This selection allows us to keep the most important observations, while still being realistic about what the patients can detect themselves. Most of the selected data can be interpreted using simple word substitution of common abbreviations and factual language. One exception is the interpretation of information given in the neutral null method. It is used to describe the mobility of the knee in the form of 0/0/129 using three degree values:

$$Hyperextension/ExtensionDeficit/Flexion$$

Table 4.5 shows the interpretation of the respective values and figure 4.6 shows an example of split and interpreted examinations.

Condition	Interpretation
Hyperextension	
0	no hyperextension of the knee
>0	hyperextension of the knee is possible
Extension deficit	
0	no extension deficit of the knee
>0	extension deficit of the knee
Flexion	
>= 120	good flexion of the knee
>= 90	adequate flexion of the knee
< 90	restricted flexion of the knee

Table 4.5: Mapping of neutral null method values to interpretation

Example 421

leg axis straight right knee joint extension/flexion no hyperextension of the knee extension deficit of the knee adequate flexion of the knee springy extension inhibition, no joint effusion, pressure pain lateral joint space

Example 1

fluent gait pattern under full load, left knee rom no hyperextension of the knee no extension deficit of the knee good flexion of the knee, opposite side hyperextension of the knee is possible no extension deficit of the knee good flexion of the knee, discrete effusion

Figure 4.6: Patient examples (examination split in observations, filtered and interpreted)

4.3.3 Free Text Generation

The final step is the generation of free text examinations. For this we are using a Mistral Model which is fine-tuned on German text[3]. To achieve better results, we are employing few-shot prompting, by giving the model two example interpretations. Since there could still be factual language in the interpreted examinations, we also instruct the LLM to replace those parts with informal language. Two examples for the free text examinations are shown in figure 4.7.

[3]https://huggingface.co/jphme/em_german_leo_mistral

Example 421

The right knee joint has extension and flexion, but it is not possible to hyperextend the knee or to fully extend it. There is an extension deficit of the knee. Flexion of the knee is only possible to a reasonable extent. There is a springy extension hematoma, no joint effusion and pain in the lateral joint.

Example 1

Under full weight-bearing, the gait pattern is fluid. The left knee cannot be flexed (hyperextension), there is no deficit in the extension of the knee. Flexion of the knee is possible. The right knee can be flexed (hyperextension), there is no deficit in the extension of the knee. Flexion of the knee is possible. There is a discrete effusion in the left knee.

Figure 4.7: Patient examples (examination rewritten)

4.4 Final Study Dataset Preparation - JK

4.4.1 Patient Summary

The input for the models was in the following format:

[AGE] years old and [GENDER]. [ANAMNESIS]. [EXAMINATION].

This comprises a summary of the patient data, similar to how the questionnaire answers would be structured.

4.4.2 Study Dataset Labels

Due to the high number of different codes (33), we decided to classify the diagnoses into subcategories. The categorization of the diagnoses for meniscus, cruciate ligament and other can be seen in the figures 4.8, 4.9 and 4.10 respectively.

For the actual training data we used the above defined labels for the prediction.

4.5 Questionnaire Design - JK

Our system design required to get the information directly from the patients, for this it is necessary to create a questionnaire that asks for all relevant information. Since we will only use the answers of the patients, and not the questions, they are phrased to encourage the patients to put all the information in their answer, instead of just yes or no.

The questions:

Urgency	ICD Code	Description
Urgent	M23.3	Possible rupture
	S83.2	Rupture
Minor	M23.0	Ganglion
	M23.1	Congenital condition
	M23.2	Old rupture issues
	M23.9	Weakness, no rupture

Figure 4.8: Meniscus Condition Classification

Urgency	ICD Code	Description
Urgent	M23.6	Rupture
	S83.50	Possible rupture
	S83.53	Rupture
	S83.54	Rupture
	S83.7	Multiple areas urgent
Minor	M23.8	Weakness, no rupture
	S83.51	Distorsion
	S83.52	Distorsion

Figure 4.9: Cruciate Ligament Condition Classification

Urgency	ICD Code	Description
Minor	M17	Gonarthrosis
	M22	Diseases of the patella
	M23.4	Free joint body
	M23.5	Instability of the joint
	S83.1	Luxation of the joint
	S83.6	Distorsion of the knee

Figure 4.10: Other Knee Conditions Classification

- Please enter your age.
- Please state your gender.
- If you had an acute event with your knee joint, what exactly happened? (e.g. twisting during sport, while walking, when straightening up from a deep squat)
- Was there anything unusual with your knee during the event, such as a twist, impact or similar?

- If you did not have an acute event with your knee joint, in what context did your pain first occur?
- Do you have any specific complaints? (e.g. swelling, pain when walking)
- Please describe the nature and location of your complaints.
- Describe the mobility of your knees, especially when stretching out.
- If you have had a knee injury in the past, what was it? (e.g. meniscus injury, injury to the anterior cruciate ligament, dislocation of the patella, cartilage damage?)

4.6 Questionnaire Dataset - JK

Our interpretation of the examinations is based on assumptions and discussions with Dr. Höher from Sportsclinic in Cologne. Nevertheless, it is possible we did not manage to achieve a realistic result of how a patient would enter the data. To test this we needed data directly entered by patients, as a ground truth to evaluate the results of the Risk Predictior against. As a mean to collect the data, we developed an iPad App to collect new data during the time of our work on this thesis. The App provides a questionnaire, whose answers the patients can enter either via keyboard or microphone. This makes it possible to already evaluate our solution on real world data. The patients entering the clinic were offered to participate while they wait for their appointment. As with the current patient dataset, we are again using the patientID to identify the patients internally. This also allows us to compare data entered by the patients with the anamneses, examinations and diagnosis later performed by the doctor.

4.7 Information Extraction - AT

Although the study dataset is in free-text format, we hypothesize that the models could instead or additionally use information structures, such as topics, keywords, entities, as input. Therefore, we explored Topic Modelling, Keyword Extraction and Named Entity Recognition for structuring the data. The best performing representatives for our purpose from each of the three algorithm types were also practically compared on the same free-form custom data entry. We chose as the final solution the representative that best met the following criteria:

1. It captures the most important phrases (not only single words) from the text
2. It does not change the information semantics, e.g. by splitting negative particles from the words they describe
3. An algorithm is treated with higher priority if it is computationally lightweight and if it is multi-language without additional tuning

4.7.1 Topic Modelling - AT

Topic Modelling identifies semantic patters within a document and common features in a text corpus. Its premise is that each document has "hidden" themes that build its meaning, called latent topics, and each of these topics consists of words, also called terms. We explored two leading computational topic models: Latent Semantic Analysis (LSA) and Latent Dirichlet Allocation (LDA).

Latent Semantic Analysis (LSA)

LSA is the foundational topic modelling technique and is advantageous for reducing highly-dimensional data, which is a common problem is NLP text data tasks (Dumais, 2004). It ingests a text corpus and creates a document-term matrix, in which each term is assigned a value, commonly a tf-idf score. Then this matrix is reduced in dimension using techniques like SVD, or singular value decomposition, which results in document vectors and terms vectors expressed in terms of topics.

Latent Dirichlet Allocation (LDA)

LDA is based on pLSA (probabilistic LSA), which instead of using dimension-reduction methods from linear algebra, like SVD, computes the probability with which a certain topic can reproduce the document-term matrix, mentioned earlier.(Pritchard et al., 2000) However, pLSA is prone to overfitting because its parameters grow linearly and cannot handle new documents because it lacks probabilities for them. LDAs variants address these limitations by using Dirichlet distributions, which do not require document sampling to find a topic, like in pLSAs, because they generate probable topic distributions instead. (Jordan et al., 2003) These advantages makes LDAs the most popular and seemingly the most effective topic modelling strategy and thereby the topic modelling representative of choice.

4.7.2 Keyword Extraction - AT

Tf-idf

Tf-idf, or Term Frequency-Inverse Document Frequency, computes a weight for the words in a text corpus by examining how often they appear in a given document compared to the whole body of texts (Jones, 1972) (Manning et al., 2009). The final score is the product of the occurrences of the word in that document divided by the number of all documents and the number of all documents divided by the amount of documents the term is present in. As a results, words are common in a specific text, but rare in the overall corpus, achieve high scores. However, Tf-idf results in singular key words rather than phrases, which contradicts our aim.

TextRank

TextRank extracts the words from a given text sample and models them and their semantic relationships as respectively the nodes and edges of a graph. Afterwards, a ranking algorithm is executed until it converges and produces stable word scores, merges words to produce multi-word entries when necessary and identifies the highest scoring entries as the keywords. (Mihalcea & Tarau, 2004) This approach tends to produce shorter keywords and is less time-efficient, which is disadvantageous.

RAKE

RAKE, Rapid Automatic Keyword Extraction, extracts keywords from a given documents without any additional context, like a text corpus. This algorithm gathers phrases by splitting the text by stop words, like "the" and "a" and counts the occurrences of every word from a phrase by itself and in combination with every other word (co-occurrence) in the document. All words from a phrase are then evaluated separately by dividing their co-occurrence counts by their overall document occurrences. In the end, the phrase scores are calculated as the sum of the scores of the words they contain. This means that words that occur commonly in a certain phrase, but not in the whole document, score higher. (Berry & Kogan, 2010)

4.7.3 Named Entity Recognition (NER) - JK

While keyword extraction focuses on selecting the most important words of a text, NER goes one step further, and determines the type of keyword found. This makes it a powerful tool for structuring data formally, but because of the labeling of the keywords, it is typically not possible for NER to be heuristic-based. Instead, it often relies on Neural Networks trained on labeled data. spaCy for NER is one of the leading NER approaches. The underlying architecture of spacy is implemented using a CNN (Vasiliev, 2020) (J. Li et al., 2020). spaCy needs to be fine-tuned for any new domain. For the medical sphere, there exist several different specialized implementation in the ScispaCy library. We choose the en-core-sci-md, since it is pretrained on biomedical data (Neumann et al., 2019).

4.7.4 Algorithm Selection

After comparing how successful the different algorithms are at extracting information structures from an example anamnesis shown in Figure 4.11, we chose RAKE as the final solution, because it most efficiently extracts keywords that summarize the key symptom from the uninterpreted anamnesis. Because of the nature of the LDA algorithm, it excludes vital information and breaks semantic relations by splitting multi-word expressions, such as fluid accumulation, which goes against criteria 1) and 2). spaCy, on the other hand, requires pre-processing and employs machine learning, which decelerate the process, and thus earns a lower priority based on criterion 3).

Spacy

1 month ago sudden knee joint blockage left knee joint. Presentation at the hospital. In the course of presentation at the hospital with X-ray and fracture exclusion. 3 years ago pre-operation. As a child already recurring blockade events. The left knee can be extended to a limited extent and flexed with difficulty. There is no fluid accumulation in the knee. There is significant pain on the outside of the knee when pressure is applied.

RAKE

1 month ago sudden knee joint blockage left knee joint. Presentation at the hospital. In the course of presentation at the hospital with X-ray and fracture exclusion. 3 years ago pre-operation. As a child already recurring blockade events. The left knee can be extended to a limited extent and flexed with difficulty. There is no fluid accumulation in the knee. There is significant pain on the outside of the knee when pressure is applied.

LDA

1 month ago sudden knee joint blockage left knee joint. Presentation at the hospital. In the course of presentation at the hospital with X-ray and fracture exclusion. 3 years ago pre-operation. As a child already recurring blockade events. The left knee can be extended to a limited extent and flexed with difficulty. There is no fluid accumulation in the knee. There is significant pain on the outside of the knee when pressure is applied.

Figure 4.11: Keywords (highlighted in red) extracted by spaCy, RAKE and LDA from an example anamnesis

5 System Design - JK

This chapter describes the solution domain. First the system structure is described in section 5.1 and secondly the development environment in section 5.2.

5.1 Deployment Structure of the system

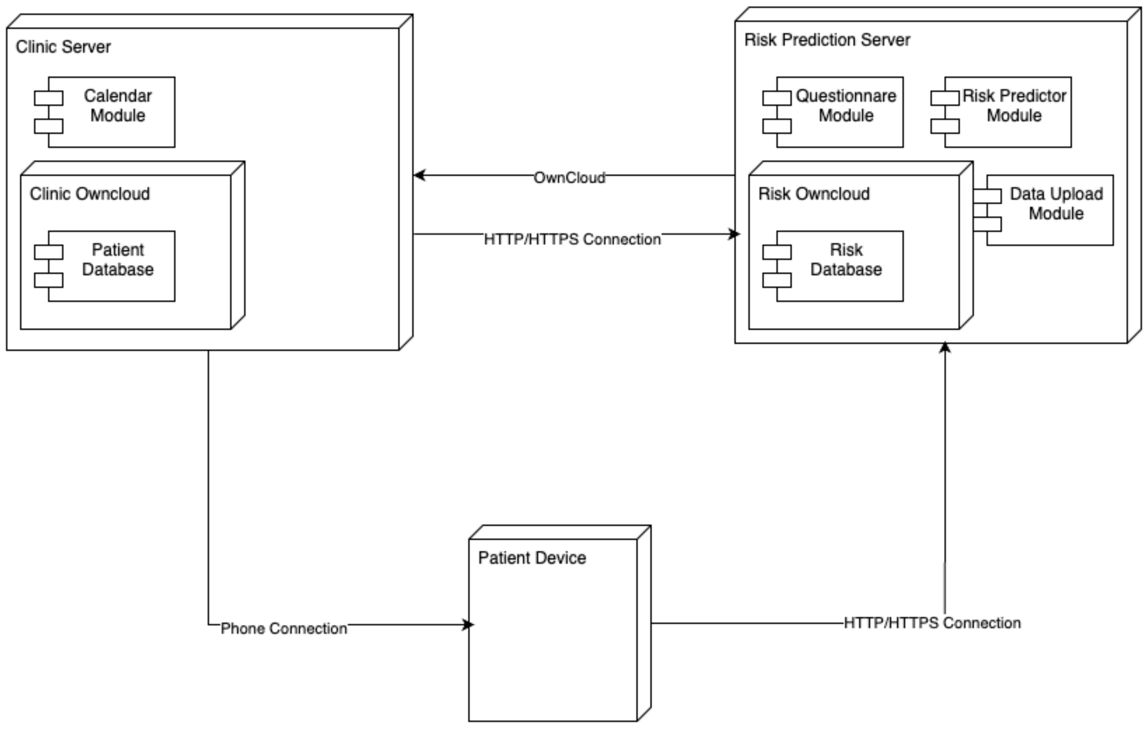


Figure 5.1: System structure using a UML Deployment Diagram

The UML deployment diagram in Figure 5.1 illustrates the architecture of the patient prioritization system, comprising three primary components: the **Clinic Server**, the **Risk Prediction Server**, and the **Patient Device**. The Clinic Server hosts a **Calendar Module**, which manages scheduling and calendar functionalities, and an **OwnCloud** instance containing the **Patient Database**, where patient information is securely stored. The Clinic Server connects to the Risk Prediction Server via an an **HTTP/HTTPS connection**, while the reverse connection from the Risk Prediction Server to the Clinic Server is handled via an **OwnCloud** protocol for secure data transfer. In the Risk Prediction Server, the **Questionnaire Module** administers and collects responses from patient questionnaires, while the **Risk Predictor Module** utilizes this data to forecast the patient risks. Additionally, the **Data Upload Module** ensures that the risk predictions and questionnaire answers are uploaded to the **OwnCloud** instance, containing the **Risk Database**. The **Patient Device**, which could be a mobile phone or other personal device, connects to the system through a **Phone Connection** to the Clinic Server and an **HTTP/HTTPS Connection** to the Risk Prediction Server.

5.2 Development Environment

The experiments of the Risk Predictors is conducted on 2023 MacBooks equipped with M3 Max Chips and 64 GB of RAM, running Sonoma 14.5.

5.2.1 Programming Language

Python was chosen as the primary programming language due to its extensive support for data science and machine learning ecosystems. Python's widespread adoption in the scientific community, coupled with its readable syntax and robust library support, makes it an ideal choice for developing machine learning models and conducting data analysis.

5.2.2 Libraries

This subsection provides details about the libraries used during the work on the thesis.

Data Handling and Processing Libraries

- **os**: Facilitates directory and path manipulations, crucial for organizing the data and model files.
- **pandas**: Provides high-level data structures and wide-ranging tools for data manipulation, enabling efficient data cleaning, transformation, and analysis.
- **sklearn**: Offers numerous tools for data preprocessing, model selection, and evaluation, essential for preparing data and tuning models.
- **numpy**: Serves as the foundational package for scientific computing in Python, offering comprehensive mathematical functions.
- **re and nltk**: These libraries support complex text processing tasks and natural language processing, which are crucial for preparing and analyzing textual data.
- **word2vec (gensim)**: An efficient implementation for training word2vec models.
- **spacy**: Provides an implementation of named entity recognition.

Deep Learning Libraries

- **tensorflow and keras**: These libraries provide comprehensive tools to build and train advanced deep learning models including LSTMs. TensorFlow offers a flexible and efficient computing system, while Keras provides a high-level API for neural network design and training. TensorFlow was utilized with hardware acceleration via TensorFlow Metal, optimizing performance by leveraging the full capabilities of the M3 Max Chips' GPU.

Transformer Libraries

- **transformers**: The transformers library, developed by Hugging Face, is a powerful tool for working with state-of-the-art natural language processing (NLP) models based on the transformer architecture. It provides access to a wide range of pre-trained models, including BERT, GPT, RoBERTa, and others, allowing easy fine-tuning and deployment for various NLP tasks such as text classification, translation, and question answering.
- **mlx**: The python library is a versatile toolkit designed for creating and managing machine learning experiments, with built-in support for hardware acceleration on Mac, providing utilities for data processing, model training, evaluation, and visualization to streamline the development workflow.

Visualization and Evaluation Libraries

- **matplotlib and seaborn**: These plotting libraries are essential for visualizing data and interpreting the behaviors of models. Matplotlib provides a wide range of plotting functions, while seaborn offers a high-level interface for drawing attractive statistical graphics.

Utility Libraries

- **pickle**: Used for serializing and deserializing Python object structures, enabling the storage and retrieval of model states.
- **datasets**: Facilitates access to a vast number of pre-processed datasets, which is crucial for benchmarking and evaluating models.
- **concurrent (multithreading)**: Utilizes multithreading capabilities to improve the efficiency of data processing and model training operations by parallelizing tasks that are not computationally dependent.
- **flask**: A lightweight Python web framework that simplifies building web applications. It's known for its flexibility and minimalism, allowing developers to start small and expand with extensions as needed.

6 Object Design

This chapter describes the solution domain. In section 6.1 we select two algorithms for the Risk Predictor, introduced in Figure 3.3 on page 15, for implementation. In the other two sections 6.2, 6.3 we examine the underlying technology of the two chosen algorithms and conduct an initial feasibility experiment. The actual implementations of the algorithms follow in chapters 7 and 8, respectively.

6.1 Risk Prediction Algorithm Selection

In this section, we evaluate different possible algorithms, to determine which ones should be investigated further.

6.1.1 Rule-Based Systems - JK

Rule-Based Systems have been around for a long time, and have already been used in the field of medicine since the 1970s (Lucas & Gaag, 1991) (Shortliffe, 1977) (Kulikowski et al., 1982) (Elkin et al., 2010). However, they require handwritten rules, supplied by medical professionals, which is their main disadvantage. This also makes it impossible to easily adapt such systems to a new medical specialty. Because of these drawbacks, we decided not to investigate an implementation of the Rule Based Predictor.

6.1.2 Self-Trained Deep Neural Networks - JK

From the plethora of different deep learning architectures, which exist, several could be used for text classification. RNNs are one of the most popular architectures for anything related to NLP. One major drawback they have, the possibility of vanishing and exploding gradients, which makes it difficult to retain any long-term dependencies. LSTM keeps an extra long-term memory of important information. This and its ability to handle highly complex data make it a perfect candidate for the risk prediction algorithm (Hochreiter & Schmidhuber, 1997).

6.1.3 LLMs - AT

LLMs can perform a variety of NLP tasks, including text classification and generation, because they not only produce a novel natural language answer, but also often understand their text input. The base knowledge that these models acquired by being trained on a large corpus of data allows to easily specialize them on a certain field and its terminology. Despite this, LLMs are underutilized in medicine, because these models could generate wrong or biased information, i.e. hallucinations. However, different means to reduce hallucinations and harmful output exist. These characteristics make LLMs suitable for diagnosis and injury risk estimation if handled correctly.

6.2 LSTM Risk Prediction Algorithm - JK

To comprehensively understand Long Short-Term Memory (LSTM) networks, it is crucial to examine their foundational components and historical developments. This section begins by exploring the origins of neural networks in sections 6.2.1 and 6.2.2, progresses through the evolution of Recurrent Neural Networks (RNNs) in section 6.2.3, leading to LSTMs in sections 6.2.4 and 6.2.5, discusses various LSTM architectures, and concludes with an initial feasibility study employing an LSTM model in section 6.2.6.

6.2.1 Biological History

In the 1930s, scientist were inspired to make machines replicate the brain's information processing (Thorndike, 1931). Grey matter is part of the brain that consists of biological neurons, which are connected to one another, building a biological neural network, in order to receive and forward electrochemical signal from and to their neighbours. These signal reach in the end receptors, which interpret the signal and process this information. That process became the foundation of artificial neural networks. (Kandel et al., 2021)

6.2.2 Artificial Neural Networks (ANN)

The basic units of neural networks are the neurons (also known as cells). As illustrated in figure 6.1 neurons are interconnected with each other using connections, each connection has a given weight, which determines its importance. Each neuron can have a number of input or output connections. Internally, a neuron has a bias and at least one activation function. The functionality of a basic neuron is as follows: First, all received inputs are multiplied with the weight of their connection. Those are then summed up and added to the internal bias of the neuron. The result of this is passed through the activation function, which returns the output of the neuron (I. Goodfellow et al., 2016) (Suzuki, 2011) (Haykin, 1998).

Mathematical expression of the neuron:

$$y(t) = F\left(\sum_{i=0}^{m} w_i(t) \cdot x_i(t) + b\right)$$

Where:

- $x_i(t)$: The input value of input i.
- $w_i(t)$ The weight for the connection i.
- b: The bias of the neuron.

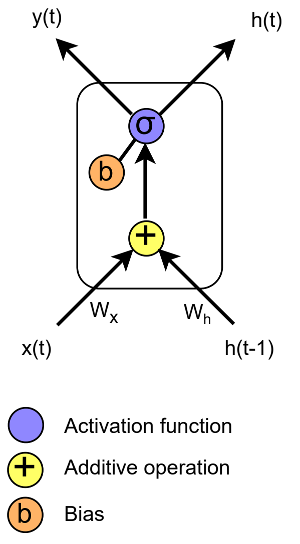


Figure 6.1: Schematic representation of a neuron (Based on (Yu et al., 2019))

- F: The activation function of the neuron.
- $y(t)$: The output of the neuron.

Neurons are often arranged in multiple layers, which are connected using the weighted connections. The simplest form of a deep neural network consists of three layers: an input layer, one hidden layer, and an output layer. The data passes through the layers of a neural network from the input layer over any hidden layers to the final output layer in a process called forward propagation. Initially, data is fed into the input layer, which passes the information to the first hidden layer. Each neuron in a hidden layer computes its output. This output is then transmitted to the subsequent layer. This process continues through each hidden layer until the data reaches the output layer. The final values of the output layer, are then the output of the neural network.

Training

During training, the output of the neural network is compared to the expected result using a loss function, which quantifies the error or discrepancy between the predicted and actual values. To achieve a better result and reduce the loss, all weights and biases are adjusted in a process called backpropagation. For each of the values, the gradient of the loss function with respect to each weight and bias is computed using the chain rule. This method allows for the efficient calculation of gradients by propagating the error backward through the network, from the output layer to the input layer (Wythoff, 1993). The computed gradients indicate the magnitude and direction the bias and weights could be adjusted to minimize the loss (Amari, 1993). The calculated gradients are then used in conjunction with an optimization algorithm, such as gradient descent, to update the weights and biases iteratively (Ruder, 2016). Such a neural network, where between neurons are unidirectional and there are no feedback loops are known as Feed-Forward Artificial Neural Networks (Suzuki, 2011).

6.2.3 Recurrent Neural Network (RNN)

Unlike feed-forward neural networks, Recurrent Neural Networks (RNNs) incorporate feedback connections, also known as back-loops. Back-loops enable RNNs to maintain and utilize information from previous time steps (words in the sentence), making them particularly effective for processing sequential data. (Yu et al., 2019)

RNN-Cell

A fundamental component of an RNN is the RNN-Cell, shown in figure 6.2. It is responsible for managing the internal state and feedback connections. When processing a sequence, such as a sentence, each token is handled sequentially, with each token corresponding to a different time step. At each time step, the cell state is updated based on the input and the previous state. For tasks like classification, typically, only the final state of the last layer is used to produce the result (Lee & Dernoncourt, 2016).

Mathematical expression of the RNN-Cell:

$$h(t) = \sigma(W_h h(t-1) + W_x x(t) + b), y(t) = h(t),$$

1. $x(t)$: Input at time step t.
2. $h(t)$: Recurrent information at time step t.
3. $y(t)$: Output at time t.
4. W_h, W_x: Weights for recurrent information and input, respectively.
5. b: Bias of the cell.

Backpropagation Through Time (BTT)

Backpropagation, which is effective for acyclic neural networks, is not suitable for training Recurrent Neural Networks (RNNs) due to their cyclic nature. Backpropagation Through Time is the training algorithm developed for neural networks using feedback connections. In this approach, the error is propagated backwards through multiple time steps to calculate the loss-gradients. Using those, the weights and biases are adjusted using a gradient descent optimization algorithm. (Werbos, 1990)

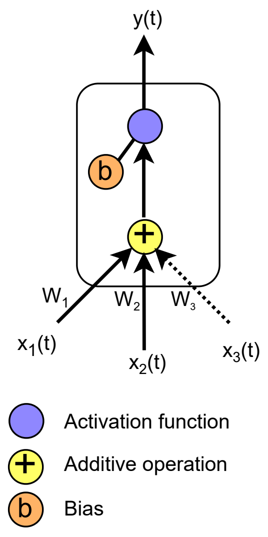


Figure 6.2: Schematic representation of a RNN-Cell (Based on (Yu et al., 2019))

Problems with long term dependencies

Despite RNNs been developed to work with sequential data, they face challenges learning long term dependencies. When the gaps between related input grows, it often comes to either of two problems:

Vanishing gradients. *Vanishing gradients* occur, when the gradients of the loss function approach 0, making it impossibly slow for the RNN to learn long term dependencies (Pascanu et al., 2013).

Exploding gradients. In contrast, *exploding gradients* occur when gradients grow exponentially during training. This can happen due to long term components growing exponentially more than short term components (Pascanu et al., 2013).

The two problems predominantly affect the weights, since during training they are multiplied with the input. The biases are only added and thus less affected by the problems.

6.2.4 Long Term Short Term Memory (LSTM)

The LSTM was developed to detect long term dependencies over more than 1000 steps (Hochreiter & Schmidhuber, 1997). This is made possible by enforcing constant error flow, with a so-called constant error carousel (CEC). The cell state carries this constant error flow backward through time, preserving the gradient and enabling the network to learn long-term dependencies.

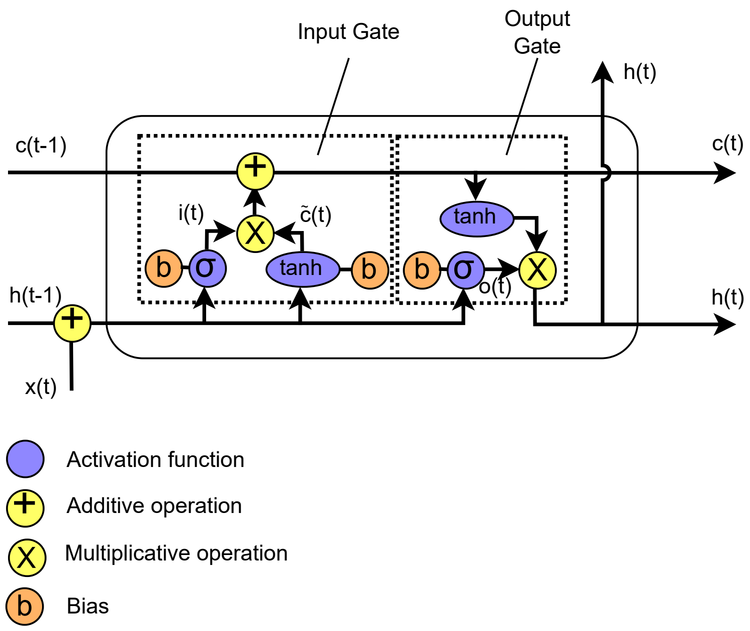


Figure 6.3: Schematic representation of a LSTM-Cell with only input and output gate (Based on (Yu et al., 2019))

Original LSTM

The first version of the LSTM cell is depicted in figure 6.3 and contained input and output gates. The input gate uses a sigmoid activation function to regulate the flow of information into the cell state. This gate determines the relevance of the incoming data, which is then scaled by a hyperbolic tangent function. The output gate, also controlled by a sigmoid function, decides how much of the cell state information is passed to the next layer. The resulting output is computed by element-wise multiplication of the sigmoid output and the hyperbolic tangent scaled cell state (Hochreiter & Schmidhuber, 1997).

Mathematical expression of the LSTM-Cell:

$$
\begin{aligned}
i(t) &= \sigma(W_{ih}h(t-1) + W_{ix}x(t) + b_i),\\
\tilde{c}(t) &= \tanh(W_{ch}h(t-1) + W_{cx}x(t) + b_c),\\
c(t) &= c(t-1) + i(t) \cdot \tilde{c}(t),\\
o(t) &= \sigma(W_{oh}h(t-1) + W_{ox}x(t) + b_o),\\
h(t) &= o(t) \cdot \tanh(c(t)),
\end{aligned}
$$

1. $c(t)$: Denotes the cell state at time t.
2. $h(t)$: Denotes the recurrent information at time t.
3. W_i, W_c, W_o: Weights associated with the input gate, cell state update, and output gate, respectively.

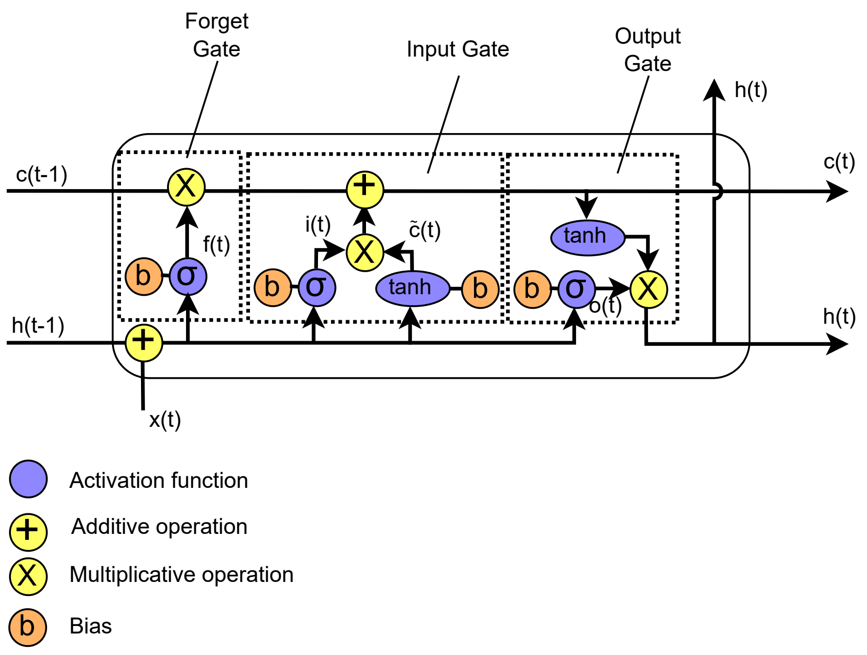


Figure 6.4: Schematic representation of a LSTM-Cell with forget gate (Based on (Yu et al., 2019))

4. b_i, b_c, b_o: Biases for the input gate, cell state update, and output gate, respectively.

LSTM with forget gate

A more recent version of the LSTM-Cell is shown in figure 6.4. It incorporates a Forget Gate and is commonly used in modern applications. The Forget Gate uses a sigmoid activation function to decide which parts of the cell state should be discarded. By applying this gate, the LSTM can effectively remove irrelevant or outdated information from the cell state, allowing it to retain and focus on more important data (Gers et al., 2000).

Mathematical expression of the LSTM-Cell:

$$\begin{aligned}
f(t) &= \sigma(W_{fh}h(t-1) + W_{fx}x(t) + b_f), \\
i(t) &= \sigma(W_{ih}h(t-1) + W_{ix}x(t) + b_i), \\
\tilde{c}(t) &= \tanh(W_{ch}h(t-1) + W_{cx}x(t) + b_c), \\
c(t) &= f(t) \cdot c(t-1) + i(t) \cdot \tilde{c}(t), \\
o(t) &= \sigma(W_{oh}h(t-1) + W_{ox}x(t) + b_o), \\
h(t) &= o(t) \cdot \tanh(c(t)).
\end{aligned}$$

1. $c(t)$: Denotes the cell state at time t.

2. $h(t)$: Denotes the recurrent information at time t.

3. W_i, W_c, W_o: Weights associated with the input gate, cell state update, and output gate, respectively.

4. b_i, b_c, b_o: Biases for the input gate, cell state update, and output gate, respectively.

Further cell structure types

There are several other variants of the LSTM-Cell:

LSTM-Cells with Peephole-Connections. This variant includes peephole connections, enabling the cell to use information about the cell state itself in the gating mechanisms. Such a structure facilitates learning more precise timing algorithms and enhances the model's ability to capture temporal dependencies (Yu et al., 2019) (Gers & Schmidhuber, 2000).

Gated Recurrent Unit (GRU). It is a simplified variant of the LSTM cell, designed to reduce computational complexity while maintaining comparable performance. It features only two gates: a reset gate and an update gate. The update gate combines the functions of the input and forget gates from the LSTM into a single gate. The reset gate determines how much of the past information to forget (Chung et al., 2014) (Yu et al., 2019).

6.2.5 LSTM Types

The different types of LSTMs can be combined with each other. This section describes examples, which will be further investigated in chapter 7.

Stacked

The simplest and most common structure is the stacking of multiple LSTM-Layers. For this structure, the output of the layer l_x at depth x serves as the input for layer $l_x + 1$. This stacking creates a deep LSTM network where each layer captures different levels of abstraction in the data (Fernández et al., 2007) (Yu et al., 2019).

Bidirectional

A bidirectional LSTM architecture incorporates two separate LSTM layers: one processes the input data in the forward direction, while the other processes it in the backward direction. This dual-layer approach enables the network to capture information from both past and future contexts, improving the overall understanding of the input sequence (Huang et al., 2015).

Attention-Based LSTM

The attention mechanism is used to enhance the model's ability to focus on the most relevant parts of the input sequence. An attention mechanism, typically implemented as a small neural network, evaluates the importance of different hidden states from various time steps based on the last hidden layer. It computes a context vector that weights these hidden states according to their relevance for the output. During training, the parameters of the attention mechanism are trained the same way as the rest of the LSTM (Niu et al., 2021) (Y. Wang et al., 2016) (G. Liu & Guo, 2019).

Convolutional LSTM (ConvLSTM)

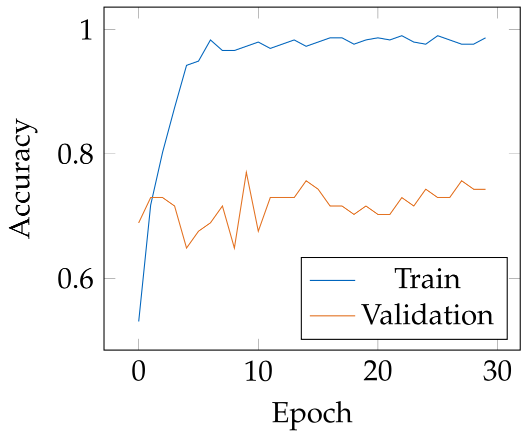
Convolutional LSTMs (ConvLSTMs) combine convolutional and LSTM layers to capture both spatial and temporal features, making them well-suited for tasks involving structured text data. In the context of text, ConvLSTMs use convolutional layers at the initial stages to extract local patterns and features from the text, such as n-grams or phrase structures. These convolutional operations help in identifying significant patterns and reducing the dimensionality of the data before it is processed by the LSTM layers (Zhou et al., 2015) (O'shea & Nash, 2015) (G. Liu & Guo, 2019).

6.2.6 Feasibility experiment

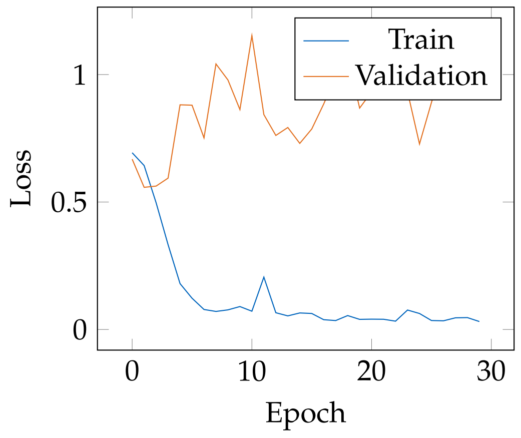
The experiment using an LSTM algorithm focused on the differentiation between general meniscus and cruciate ligament injuries, by classifying patient anamneses. In this initial dataset there were more cruciate ligament anamneses available and to prevent any model bias, the anamneses were undersampled for an even distribution between the two diagnoses. For each type of diagnosis, 184 entries were used. 20% of the 368 anamneses were reserved as the validation data.

For the actual training, the anamneses needed to be in a machine-readable format. Therefore, they were transformed into lists of 100 numbers between 0 and 10,000 using tokenization and padding. They were then fed into the embedding layer of the model, which maps them to a dense vector space with a dimension of 128. The next part of the structure contains two bidirectional LSTM layers (128 and 64 cells respectively) separated by a Dropout layer. The Dropout layer sets half the inputs to 0, removing part of the information, and hence reducing the risk of overfitting. The result of the second bidirectional LSTM is then transformed into a binary classification result. The loss of the training is calculated using categorical cross entropy. The model was trained for 30 epochs with a batch size of 32.

The training graphs can be seen in figure 6.5. The model managed to achieve a 74% accuracy on the validation data. The loss, on the other hand, concluded at 0.92. 6.5 Despite the model achieving adequate results for one of the first experiments, it also shows clear signs of overfitting. The accuracy of the model for the training data reached a considerably higher value, which shows that the model was not able to generalize well to the unseen validation data. The loss for the training data approaches 0, while the loss for the validation data shows the opposite trend, which indicates a vast difference in performance between the two sets of data. In conclusion, the experiment was definitely a success, although one with a lot of need and potential for improvement.



(a) Model Accuracy



(b) Model Loss

Figure 6.5: Feasibility-Model Accuracy and Loss during Training

6.3 LLM Risk Prediction Algorithm - AT

This section starts with an introduction to the transformer structure in subsection 6.3.1, this is then extended to the general LLM structure in subsection 6.3.2. In section 6.3.3 an initial feasibility experiment is conducted with the LLM Risk Predictor.

6.3.1 Transformer Structure

Tokenizer. *Tokenizer* is a pre-processing tool that splits textual input in discrete components, called "tokens". These tokens can represent words, sub-words or characters, which are given an ID that is then embedded for the model (Transformer or LLM) to digest. These tokens become the vocabulary, which the model uses to predict the output. Word tokenization, however, handles “out-of-vocabulary” instances (words it has never seen before) with difficulty. Character tokenization addresses this issue because the model has most likely familiar with the characters even of unknown words, but it has problems with text abstraction, as it does not capture semantic relationships in the language. Thereby, sub-word tokenizaton combines the advantages of the two other approaches. Two common sub-word tokenization techniques are BPE (Byte-Pair Encoding), WordPiece and SentencePiece. BPE creates a set of unique words with their corresponding frequency in the text corpus, then parses these words into unique symbols that become its base vocabulary. Finally, the BPE learn how to merge these symbols, e.g. by counting the which symbols occur commonly together, to build a vocabulary of a size, determined by a hyperparameter (Sennrich et al., 2015). WordPiece functions similarly, but instead of choosing the most common symbol pairs, it calculates how many words it would lose if it were to merge two symbols and aims to minimize that value (Schuster & Nakajima, 2012). Lastly, although BPE and WordPiece assume that all languages separate their words with spaces, this is not always the case, for example with Chinese. For that reasor, SentencePiece enforces space separator when parsing symbols and only then does it employ other algorithms, like BPE (Kudo & Richardson, 2018).

As mentioned, LLMs are based on the Transformer architecture. The Google Brain team developed transformers (Figure 6.6) to consist of an Encoder (on the left), which consumes the input, pre-processed using a tokenizer, and transforms it into an intermediate representation of the same size, and a Decoder (on the right), which uses the intermediate representation to generate the final input. (Vaswani et al., 2017)

Encoder

In more detail, before entering the Encoder the tokens (which come from the initial user input) are turned into vectors using an **Input embedding** layer and each tokens absolute position in the sequence is additionally added using **positional encoding**, because Transformers do not inherently understand sequential data. The body of the Encoder consists of 6 layers of **Multi-Head Attention** and **Feed Forward** sublayers one after another, with each sublayer having an additional residual connection and normalization (**Add and Norm**). Models employ attention mechanisms to determine how important the parts of input sequence (Key-value pairs K - V) are to a part of it (Query Q) to produce an output. The algorithm calculates the final output by summing the weighted values, where the weights represent how compatible a key is to a query. Multi-head attention, i.e. cross-attention, in specific uses several self-attention mechanism in parallel (called heads) to be able to focus on different parts of the input simultaneously. The residual connection adds the output of the previous sublayer to the output of the current sublayer. The current sublayer's output is a function of the previous sublayer's output. Normalization then stabilizes this result by shifting it non-strictly between -1 and 1, meaning the final output is $Normalization(x + Sublayer(x))$, where Sublayer(x) is the function implemented by the current sublayer. Feed Forward is a fully-connected sublayer, which transforms each token non-linearly two times. The first transformation captures more data features by expanding it dimensions, then follows a ReLU activation function and finally the data is again reduced to its original size to be compatible with the following layers. For each token of the input sequence, the Encoder outputs one vector that captures its context. (Vaswani et al., 2017)

Decoder

The Decoder has a separate input, which is initially only a Start-of-Sequence (marked as <s>) and by iterating through the Decoder a new token is generated and appended to the Decoder input for the next iteration, which produces the token after that. Similarly to the Encoder, the Decoder inputs are embedded and positionally encoded. The Decoder comprises a stack of 6 layers, where each includes **Masked Multi-Head Attention**, Multi-Head Attention and Feed Forward sublayers. The only sublayer that is was not present in the Encoder, but is in the Decoder is the Masked Multi-Head Attention. Additionally to the attention mechanism properties, masks the tokens after a given token, in order for the model to predict it based only on the previous tokens. Then, the Multi-Head Attention sub-layer processes the Decoder input by also considering the context for all tokens of the user prompt, which is the output of the Encoder. Then the Feed Forward sublayer transmits its output to the next Decoder layer. After all decoder layers process, a Linear layer followed by a Softmax compute the next token's probability and selects the word with the highest probability from the vocabulary. This output token is appended to the previous Decoder input and the whole next-token generation process is repeated until the model produces an End-of-Sequence token (marked as </s>). (Vaswani et al., 2017)

The decoding process could be illustrated with the following example. If thew user prompts the Transformer to translate the sentence "This is a dog" from English to German, the Encoder will generate an output $OutputE$ equal to $[Encoding_{This}, Encoding_{is}, Encoding_{a}, Encoding_{dog}]$ and the Decoder input will be <s>. Using:

- $OutputD_i$ - Decoder output from layer i
- $EmbEnc$ - Embedding and Positional Encoding
- $MaskedAttn_i$ - Masked Multi-Head Attention on layer i
- $Attn_i$ - Multi-Head Attention on layer i
- $FeedForward_i$ - Feed Forward on layer i
- $AddNorm$ - Add & Norm
- $LinearSoftmax$ - Linear and Softmax
- $argmax$ - choosing the value with highest probability

Then the first new token (*Das* as a translation of *This*) is generated as follows:

$$< s > \xrightarrow{EmbEnc} MaskedAttn_1 \xrightarrow{AddNorm+OutputE} Attn_1 \xrightarrow{AddNorm} FeedForward_1 \xrightarrow{AddNorm} OuputD_1$$

$$OuputD_1 \rightarrow MaskedAttn_2 \xrightarrow{AddNorm} ...FeedForward_6 \xrightarrow{AddNorm} LinearSoftmax \xrightarrow{argmax} Das$$

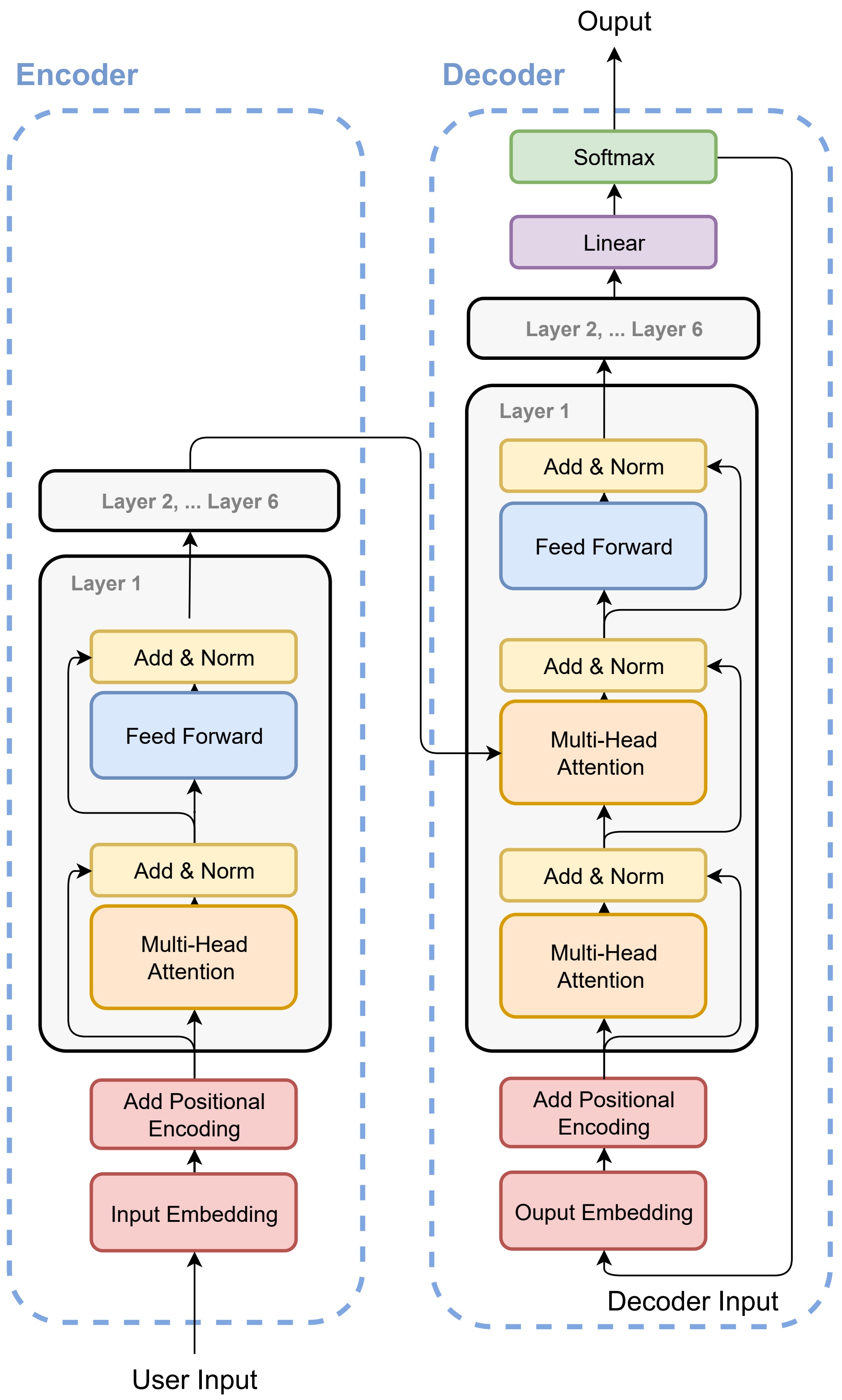


Figure 6.6: Layer structure of a Transformer (based on Vaswani et al., 2017)

6.3.2 General LLM Structure

Although some LLMs employ both of these components, most use stacks of only one of them. Encoder-only models like BERT excel in understanding the context and meaning and therefore specialize in NLP tasks, such as named entity recognition and classification. (Devlin, 2018) On the other hand, decoder-only LLMs, for example GPT and Llama, are best at generating text (Openai et al., 2018). Encoder-decoder combine the strengths of the former two types, however, their training proves to be really expensive. (T. Wang et al., 2022) Additionally, LLMs vary in size based on how many **parameters** they have, where the parameters are the variables a model learns during training and include among others weights and biases. They shape LLMs' natural language understanding and functioning, thus more parameters lead to a better performance. LLMs also possess **hyperparameters**, which are different from parameters because these variables are not computed by the model, but are manually set by developers before training. (Arnold et al., 2024) Hyperparameters determine how well the model trains and performs on unseen-before data, for example LLM temperature controls the randomness of generated output and therefore changes how deterministic or creative a model is. (Peeperkorn et al., 2024)

Fine-tuning. *Fine-tuning* adapts a pre-trained LLM to perform specific tasks or improve its performance on general tasks by training it further on a tailored dataset, which adapts the LLM's parameters, i.e. weights. This process enhances the model's ability to generate more accurate, contextually appropriate, and human-like responses and is useful for interdisciplinary projects, which require knowledge of a special field's jargon, like medicine. (Quinn et al., 2020)

Prompt. A *prompt* is a natural language text given to LLMs that describes what task they have to perform. There are user and system prompts. *User prompts* are the queries that users make while interacting with the LLM in order to get a particular response. *System prompts* are instructions that developers give to their LLM systems to provide them with context and guide them. (Radford et al., 2019a)

Hallucination. An LLM *hallucinates* when it generates responses that are factually wrong, illogical or disconnected from the prompt. LLMs inherently hallucinate, because they cannot possibly learn all the correct patterns and facts from the vast dataset that they have been trained on. However, different Prompt Engineering techniques could mitigate these hallucinations. (Xu et al., 2024)

Prompt Engineering. *Prompt Engineering* is the process of designing and crafting the LLMs' prompts to optimize them and elicit a desired response. There are multiple methodologies for textual input and output (text-to-text), the most popular being:

- Zero-shot prompting: This prompting technique relies on the pure ability of the model without providing any additional context or examples.
- Few-shot prompting: In this approach a few sample inputs and their corresponding

optimal outputs are provided in the system prompt. This technique steers the model in a wanted direction and helps it reason better its answers. (Semnani et al., 2413)

- Chain-of-Thought: With this method LLMs are encouraged to reason their decisions step by step, which results in a more coherent and correct final answer. (Wei et al., 2022)

6.3.3 Feasibility experiment

Llama 3 is the latest version of the Llama LLM. Similar to the first generation model, it has a decoder-only architecture (More in Section 8.1.2 *LLM Structure Selection* and Figure 8.4), but it additionally adopts a grouped query attention (GQA) mechanism and has a more efficient tokenizer with a 128K vocabulary. [1]

To prove that LLMs can predict the risk level of orthopedic injuries, Med Llama 3 [2] was used on a small data sample. Med Llama 3 is a Llama 3 model, which fine-tuned on English medical questions and answers (Med Q&A) without any architectural changes and q5 quantization. This fine-tuning drastically enhances the model's medical knowledge base, but only in English. For that reason, the experiment dataset was also translated to English. Additionally, this dataset follows the *Simplified user testing* principle of Discount Usability Engineering, because it consists of only five patient entries. Each of those consists of fully detailed anamneses and examination interpretations, simulating patient description of their symptomatic. (Nielsen, 1989) For the further implementation of the risk prediction algorithm, LLMs capable of German are to be investigated in the *LLM Implementation* chapter.

In this experiment, the system prompt utilizes a few-shot prompting technique that samples two actual meniscus tear patient entries and one artificial knee arthrosis entry. The residual three entries are used as validation data. In Figure 6.7, there is an example system prompt, which is as identical as possible to the one used in the experiment without revealing any real medical patient data.

As shown in the example in Figure 6.8, the model is able to give a basic diagnosis and suggest a treatment type. Table 6.1 shows that on the validation data it achieved an average of 84% cosine similarity, which means that the details in the prediction could be further increased. Additionally, the dataset was not exhaustive and biased towards meniscus bucket-handle tears, which leads to subsequent skewed results. Proper testing on more clinical data follows in chapter 9 Evaluation.

[1]https://ai.meta.com/blog/meta-llama-3/
[2]https://huggingface.co/ruslanmv/Medical-Llama3-8B

System prompt

You are an AI Medical Assistant with specialized training in a comprehensive health information dataset. Your task is to provide a precise diagnosis of the knee injury and whether the treatment for this injury is operative or conservative. Base your response solely on the provided information, without recommending additional tests or evaluations. Do not sign your answer as a doctor.

Example Input:
2 months ago sudden knee joint blockage left. Presentation at hospital with X-ray and exclusion of fracture. Previous operation 2 years ago. Already recurrent blockade events as a teenager. Left knee has limited extension and is flexed with difficulty. There is no fluid accumulation in the knee. There is significant pain on the outside when pressure is applied.

Example Output:
Post-dislocated external meniscus bucket-handle tear left, operative

Example Input:
on 15.05.23 got stuck in the grass and twisted my right knee while playing soccer, no contact with an opponent. Restricted movement. X-ray was done and MRI is upcoming. Currently continued extension deficit. The right knee can be extended to a limited extent and flexed well. No fluid accumulation in the knee. There is significant pain when pressure is applied to the inner side.

Example Output:
Right meniscus bucket-handle tear, operative

Example Input:
Since 7 months, knee pain exacerbated by walking, with relief when at rest. The knee shows marked swelling and restricted extension, preventing full straightening of the leg. No prior knee injury. Decreased range of motion and audible grinding.

Example Output:
Knee arthrosis, conservative

Your turn:

Input:

Figure 6.7: System prompt from feasibility experiment using an LLM

Diagnosis given by doctor	**Diagnosis predicted by Med Llama 3**	**Cosine similarity**
Right displaced lateral meniscus bucket-handle tear, operative	Right meniscus bucket-handle tear, operative	87%
Acute left medial meniscus bucket-handle tear, operative	Left meniscus bucket-handle tear, operative	72%
Right medial meniscus bucket-handle tear, operative	Right meniscus bucket-handle tear, operative	93%

Table 6.1: Feasibility experiment results with Medical Llama 3

Example

USER: Sat cross-legged on 08.04.2024, then began cracking and subsequent extension inhibition. Presented in hospital with X-ray and exclusion of fracture. MRI diagnosis on Tuesday with diagnosis of an external meniscus tear. For approx. 2 years, slight blockages triggered again by stretching. The right knee can be extended to a limited extent and cannot be flexed as well. There is significant pain when pressure is applied to the outer side.

Med Llama 3: Right meniscus bucket-handle tear, operative

Figure 6.8: Example from feasibility experiment using an LLM

7 LSTM Implementation - JK

This chapter explains the process undertaken to construct the optimal implementation of the LSTM Algorithm for risk prediction. First, a baseline model is established in section 7.1. It follows the evaluation of different optimization strategies against the baseline in sections 7.2, 7.3 and 7.4. At the end, in section 7.5 a final model including the useful optimization strategies is proposed.

7.1 Establishing Baseline Model

To evaluate if certain optimizations actually enhance the results of the LSTM, it is necessary to establish a baseline to compare against. All experiments are conducted on the study dataset introduced in chapter 4 and evaluated using the metrics accuracy, precision, recall, macro F1 score, and weighted F1 score. The metric which we are trying to optimize is the macro F1 score, since it is optimal for evaluating the classification performance on imbalanced datasets. To evaluate the approaches fairly, certain parameters were fixed. The following values are used for all following experiment, if not stated otherwise:

1. Epochs: 15 - chosen to balance training time and model performance.
2. Batch size: 32 - a common choice providing stable gradient estimates.
3. Adam optimizer with learning rate 0.001 – selected for its efficient handling of sparse gradients.
4. Categorical cross entropy loss – appropriate for the multi-class classification problem at hand.

For the baseline LSTM three different structures, as shown in figure 7.1, were considered:

1. Simple LSTM structure
2. Feasibility experiment with Bi-LSTM structure
3. Simple Bi-LSTM structure

Table 7.1 summarizes the evaluation results for the baseline options. For the further experiments, the simple Bi-LSTM 7.1c is the best choice, since it is not complex and still achieves the best results. The simple LSTM only predicts one of the classes, as can be seen in the confusion matrix 7.2a. One possible explanation is the class imbalance of the study dataset. The clear advantage of the Bi-LSTM leads to the conclusion, that the normal LSTM should not be considered further.

model	*accuracy*	$precision_w$	$recall_w$	$f1_w$	$f1_{mac}$
simplified Bi-LSTM	0.67	0.67	0.67	0.67	0.58
feasibiity experiment Bi-LSTM	0.65	0.65	0.65	0.65	0.53
simple LSTM	0.42	0.17	0.42	0.25	0.12

Table 7.1: Comparison of baseline possibilities

7.2 Parameter Optimizations

In this section, several optimization strategies pertaining to the input and parameters given to the model for training are evaluated. These parameters typically include the training data, number of epochs, batch size, learning rate, and more. By optimizing these parameters, the aim is to enhance the model's performance, reduce overfitting, and improve training efficiency.

7.2.1 Early Stopping

Early stopping is a technique used to halt model training when performance reaches a plateau, preventing further improvements. This method optimizes the number of epochs by stopping training once the validation performance ceases to improve. The main benefits of early stopping include reducing overfitting risk and decreasing computation time, as the model is not trained beyond its optimal point (Prechelt, 2000).

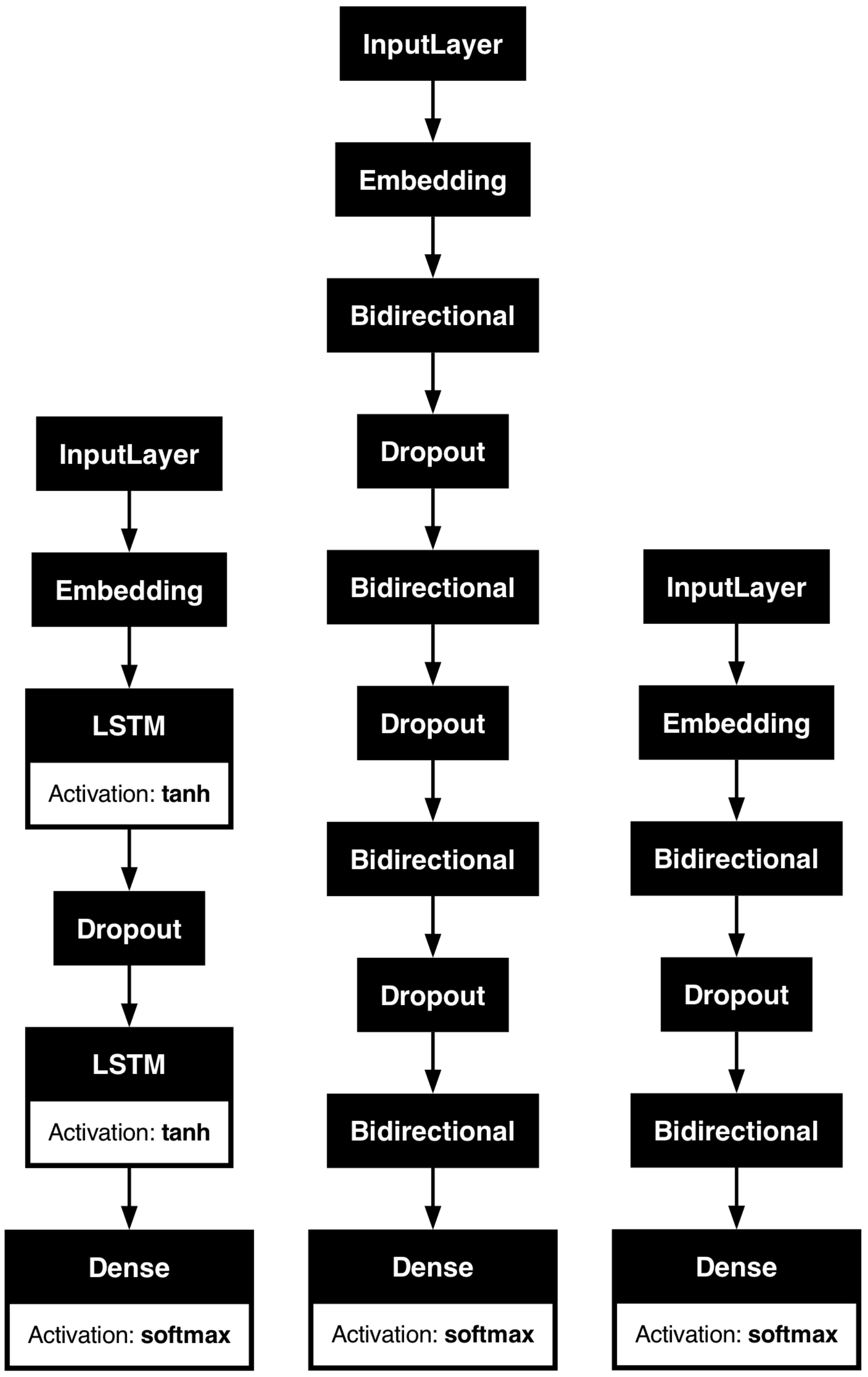
The early stopping is implemented as follows:

1. Stop the training if the model does not improve for 3 epochs by at least 0.1 in the validation macro F1 score, thereby not showing meaningful improvements
2. When stopping, the weights of the best epochs are used for the result

Since early stopping only enhances model performance and facilitates the evaluation of subsequent strategies at their best, it will be employed in all following evaluations.

7.2.2 Hyperparameter Tuning

Neural networks have numerous parameters that need optimization to achieve the best model performance. Hyperparameter tuning involves using an optimizer to explore various configurations intelligently. Bayesian Optimization is one such approach, which efficiently searches the parameter space to find optimal values (Wu et al., 2019). For this experiment, the following parameters were evaluated, and their optimal values determined:



(a) Simple LSTM structure

(b) Feasibility experiment with Bi-LSTM structure

(c) Simple Bi-LSTM structure

Figure 7.1: Comparison of structures of baseline possibilities

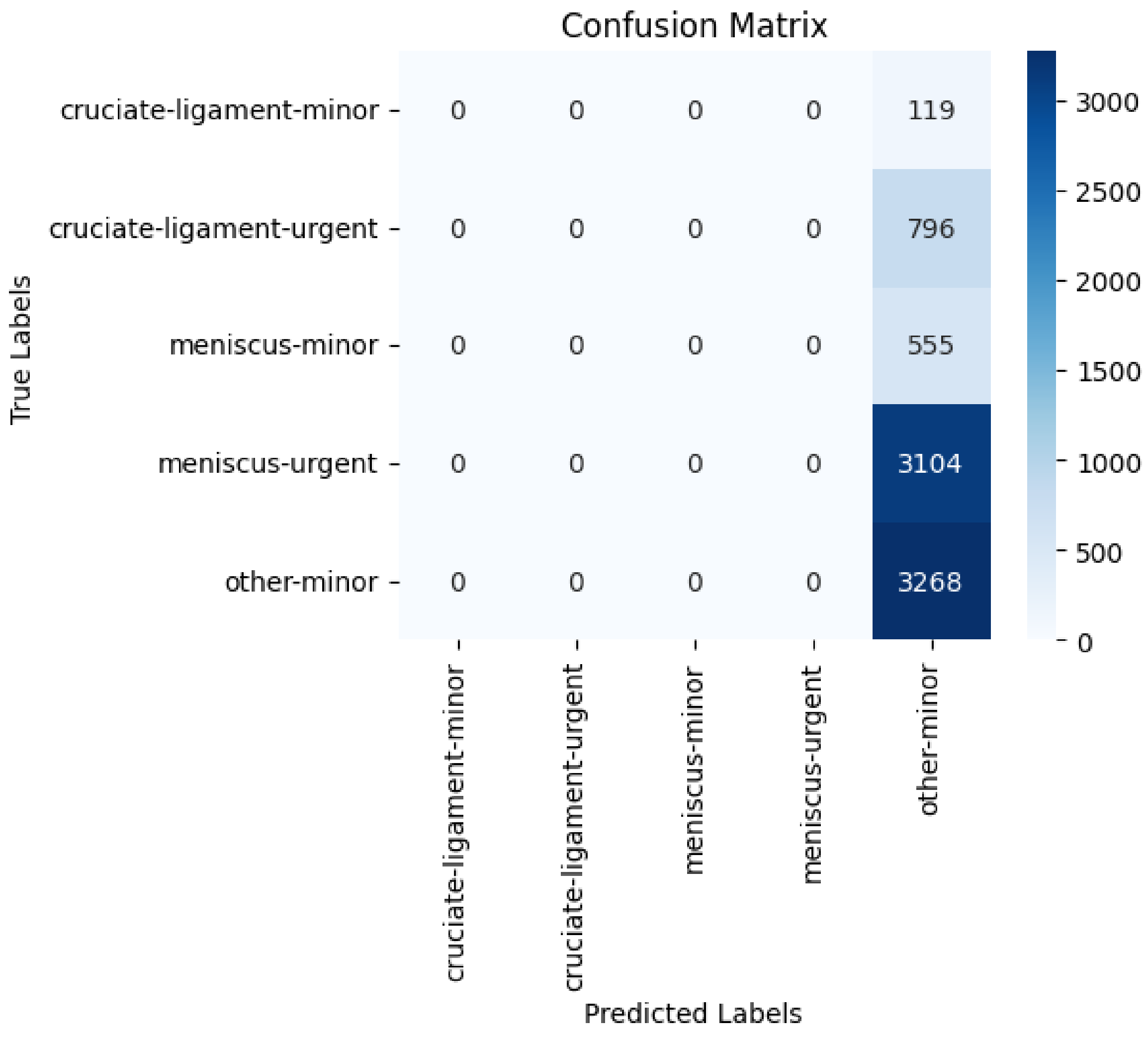


Figure 7.2: Confusion Matrix of the Simple LSTM, showing that it only predicts one class, making it unsuitable for further evaluation

1. Bi-LSTM Layer 1 units: 512
2. Bi-LSTM Layer 2 units: 448
3. Dropout: 0
4. Batch size: 64

During hyperparameter tuning, the model was trained for 9 epochs in each of 10 different trials to identify the optimal configuration.

Baseline - Tuner Comparison

The results of the hyperparameter tuning are summarized in Table 7.2, which demonstrates the performance improvements achieved through hyperparameter tuning.

model	$accuracy$	$precision_w$	$recall_w$	$f1_w$	$f1_{mac}$
baseline	0.67	0.67	0.67	0.67	0.58
tuner	0.7	0.7	0.7	0.7	0.62

Table 7.2: Performance Comparison Between Tuner and Baseline Models

The results indicate that hyperparameter tuning can significantly improve model performance. Notably, the optimal configuration completely eliminates dropout, which will be further evaluated in section 7.3.4. This suggests that, for this particular dataset, dropout may not be necessary and could potentially hinder model performance.

7.3 Training Optimizations

In this section, several optimization strategies for the training process of the model are evaluated. These include learning rate scheduling techniques, dataset balancing methods, batch normalization, regularization techniques, and class weighting strategies. By optimizing these aspects, we aim to enhance the model's performance, reduce overfitting, and improve training efficiency.

7.3.1 Learning Rate Scheduler

While hyperparameter tuning focuses on finding the optimal value for a parameter, another option is to adapt the parameter during training. One such approach is the learning rate scheduler.

Exponential Decay

Exponential decay reduces the learning rate at fixed intervals during training. This technique helps the model converge more smoothly by decreasing the learning rate as training progresses. The learning rate is typically reduced by a factor at each interval, ensuring that the model does not overshoot the optimal parameters (Z. Li & Arora, 2019).

Cyclical Learning Rate

Cyclical learning rate is an approach where the learning rate cyclically varies between a minimum and maximum value during training. This method helps the model escape local minima and potentially find better solutions (Smith, 2017). The parameters for the cyclical

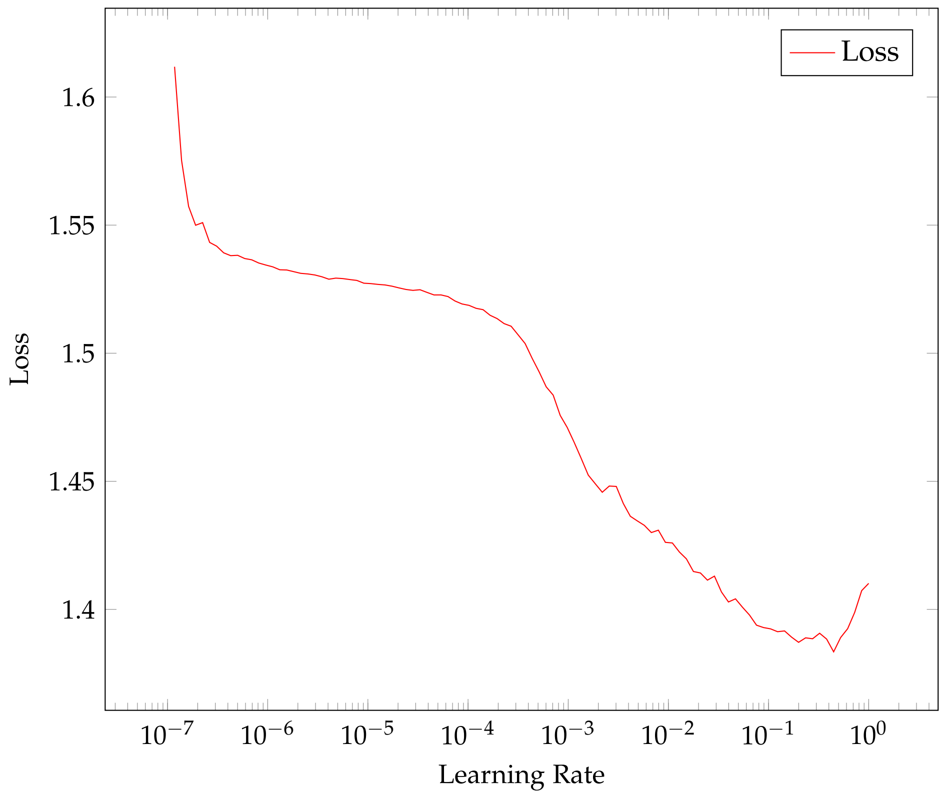


Figure 7.3: Graph detailing changes in Loss in relation to the Learning Rate

learning rates are determined based on the loss graph (Figure 7.3), by determining the minimum value as the point of steepest descent to the global minimum and the maximum as the value in the global minimum just before the loss increases again. In this case, the minimum was determined as $10^{-3.5}$ and the maximum as 0.2.

Baseline - Learning Rate Scheduler Comparison

The results in Table 7.13 show the performance comparison between the baseline model and the baseline model using a cyclic learning rate scheduler and using a decay learning rate scheduler.

model	$accuracy$	$precision_w$	$recall_w$	$f1_w$	$f1_{mac}$
baseline	0.67	0.67	0.67	0.67	0.58
cyclic learning rate scheduler	0.42	0.34	0.42	0.36	0.18
decay learning rate scheduler	0.69	0.69	0.69	0.69	0.61

Table 7.3: Impact of Learning Rate Schedulers on Model Performance

Table 7.3 indicates that the cyclical learning rate does not perform as well as expected. However, with further parameter tuning, its performance might be improved. The exponential decay, on the other hand, does improve the performance.

7.3.2 Dataset Optimization

The classes in the study dataset are not distributed equally, this could lead the model to favor the more common one, since it could already achieve a measure of accuracy by focusing on them. To alleviate this problem, it is possible to balance the study dataset.

Undersampling

One such strategy is undersampling. For this, the amount of data points for the smallest class is calculated, then only this amount is kept for any of the other classes, to allow for an even distribution (Mohammed et al., 2020a). One downside is the reduction of training data, which is especially critical for data dependent approaches such as neural networks. For the study dataset this results in retaining only 355 samples per class.

Oversampling

Oversampling involves duplicating the less common classes until an even distribution is achieved (Mohammed et al., 2020a). It is important to perform oversampling after the dataset is split into training, validation, and test sets to avoid unrealistic results. Oversampling can still hinder the model's generalization ability by making it too focused on duplicated data (Drummond, Holte, et al., 2003).

Baseline - Undersampling - Oversampling Comparison

The results in Table 7.13 show the performance comparison between the baseline model trained on the original data, undersampled data and oversampled data.

model	*accuracy*	$precision_w$	$recall_w$	$f1_w$	$f1_{mac}$
baseline	0.67	0.67	0.67	0.67	0.58
undersampling	0.35	0.45	0.35	0.37	0.29
oversampling	0.61	0.63	0.61	0.62	0.58

Table 7.4: Effect of Sampling Techniques on Model Performance

Table 7.4 indicates that the baseline model, trained on unaltered data, performs the best overall. Although oversampling achieves the same macro F1 score (0.58) as the baseline, it results in a lower accuracy (0.61 vs. 0.67), suggesting that while class balance improves representation, it may also introduce noise due to duplicated samples. On the other hand, undersampling significantly reduces performance across all metrics, with a macro F1 score of 0.29, indicating that the loss of training data severely impacts the model's effectiveness.

7.3.3 Batch Normalization

Batch normalization normalizes the output of each layer, reducing the chance of exploding and vanishing gradients, and making the training process smoother. This technique also reduces the training time required for the model to converge. The parameters of batch normalization are learned by the model, which minimizes the risk of losing information during normalization (Ioffe & Szegedy, 2015) (Cooijmans et al., 2016).

Baseline - Batch Normalization Comparison

The results in Table 7.5 show the performance comparison between the baseline model and the baseline model using batch normalization.

model	*accuracy*	$precision_w$	$recall_w$	$f1_w$	$f1_{mac}$
baseline	0.67	0.67	0.67	0.67	0.58
batchnormalization	0.68	0.68	0.68	0.67	0.6

Table 7.5: Comparison of Batch Normalization on Model Performance

The results indicate that batch normalization improves model performance, confirming its effectiveness in stabilizing the training process.

7.3.4 Regularization Techniques

The main purpose of regularization techniques is to reduce the risk of overfitting the model. We investigate the 4 different techniques: Dropout, L1 Regularization, L2 Regularization and Elastic Net.

Dropout

Dropout is typically implemented using Dropout Layers, which delete a given percentage of the input and forward the result to the next layer. This prevents the model from overfitting by ensuring it does not become overly reliant on any single feature (Baldi & Sadowski, 2013).

L1 Regularization

L1 regularization adds a penalty proportional to the absolute value of the weights. This encourages sparsity in weights, resulting in some weights being zero, thus simplifying the model and preventing overfitting (Schmidt et al., 2009).

L2 Regularization

L2 regularization adds a penalty proportional to the square of the magnitude of the weights, which generally keeps the weights smaller and helps in preventing overfitting (Cortes et al., 2012).

ElasticNet

Elastic Net combines both L1 and L2 regularization, promoting both sparsity and small weights, thereby balancing the benefits of both techniques (Zou & Hastie, 2005).

Baseline - Dropout - Recurrent Dropout - L1 - L2 - L1 + L2 Comparison

The performance comparison of different variations of the baseline model, including no dropout, using two additional dropout layers, and using either or both L1 and L2 regularization, is shown in Table 7.6.

model	$accuracy$	$precision_w$	$recall_w$	$f1_w$	$f1_{mac}$
baseline	0.67	0.67	0.67	0.67	0.58
no dropout layer	0.69	0.69	0.69	0.69	0.6
two dropout layers	0.67	0.67	0.67	0.67	0.57
l1 regularization	0.42	0.17	0.42	0.25	0.12
l2 regularization	0.58	0.58	0.58	0.57	0.43
l1 + l2 regularization	0.42	0.17	0.42	0.25	0.12

Table 7.6: Impact of Various Regularization Techniques on Model Performance

The evaluation results presented in Table 7.6 reveal notable differences in the effectiveness of various regularization techniques, especially when focusing on the macro F1 score. The baseline model achieved a macro F1 score of 0.58, while the model with less dropout layers improved slightly to 0.60. In contrast, using more dropout layers slightly reduced the macro F1 score to 0.57. L1 and L2 regularization methods, both individually and in combination

(Elastic Net), significantly underperformed in terms of the macro F1 score, with both L1 and L1 + L2 regularizations scoring as low as 0.12. L2 regularization alone managed a macro F1 score of 0.43, which is still below the baseline. Overall, the results indicate that while dropout can provide marginal improvements, excessive regularization with L1 or L2 significantly degrades performance. Therefore, a balanced approach with careful tuning of dropout layers appears to be the most effective strategy for this particular model setup.

7.3.5 Class Weights

One option to mitigate the impact of unevenly distributed datasets is to assign more importance to less common prediction classes. This can be achieved by using class weights relative to the distribution of data, thereby evening out the impact of different prediction classes (Haixiang et al., 2017).

Baseline - Class Weights Comparison

The performance comparison between the baseline model and the baseline model using class weights is shown in Table 7.7.

model	$accuracy$	$precision_w$	$recall_w$	$f1_w$	$f1_{mac}$
baseline	0.67	0.67	0.67	0.67	0.58
class weights	0.59	0.63	0.59	0.6	0.51

Table 7.7: Evaluation of Class Weights on Model Performance

The results presented in Table 7.7 indicate that the baseline model performs better without the use of class weights. The baseline model achieves higher scores in accuracy, precision, recall, and F1 metrics compared to the model using class weights. This suggests that the baseline model is inherently capable of handling class imbalance more effectively without the need for additional weighting adjustments.

7.4 Structural Optimizations

In this section, the structural optimizations are evaluated, meaning actual changes to the structure of the LSTM, which are not mainly benefitting better training.

7.4.1 Amount of layers

The number of layers used in constructing the LSTM is a crucial consideration. While more layers can capture complex patterns in the data, they also increase the risk of overfitting. Conversely, fewer layers may generalize better but might struggle with learning intricate dependencies.

Baseline - More Layers - Less Layers Comparison

The results in Table 7.8 show the performance comparison between the baseline model and the baseline model with one layer and the baseline model with three layers.

model	*accuracy*	$precision_w$	$recall_w$	$f1_w$	$f1_{mac}$
baseline (two layer)	0.67	0.67	0.67	0.67	0.58
one layer	0.71	0.7	0.71	0.7	0.62
three layers	0.67	0.67	0.67	0.66	0.56

Table 7.8: Comparison of Model Performance Based on Layer Quantity

The results in table 7.8 demonstrate that a model with fewer layers performs better than both the baseline model and the model with more layers, particularly in terms of F1-Score and Accuracy. This suggests that a simpler architecture may be more effective in avoiding overfitting and generalizing well to new data. However, it is important to recognize that this trend could vary based on the complexity of the model. Therefore, the optimal number of layers should be re-evaluated with the final model to ensure the best possible performance across different scenarios.

7.4.2 Embeddings

The embeddings are an important aspect for any neural network, especially for LSTMs, which are not pretrained on a big corpus of data. To alleviate this disadvantage, it is possible to use a pretrained embedding model.

Word2Vec

The word2vec algorithm was trained using a collection of german wikipedia data [1], to allow for a general representation of the language. It was trained using the skip-gram training algorithm. The window, maximum distance between predicted and current word, is set to 8. All words with less then 5 occurrences are removed, to not add any noise to the data. With this setup, the model was trained on the data for 10 epochs.

BERT

Another option for embeddings is using the last hidden state of LLMs. For this experiment GerMedBERT[2], is chosen, since it is fine-tuned on medical data and as such should have a more accurate representation of the terminology in the field.

[1]https://huggingface.co/datasets/legacy-datasets/wikipedia
[2]https://huggingface.co/GerMedBERT/medbert-512

Baseline - Word2Vec - BERT Embeddings Comparison

The results in Table 7.9 show the performance comparison between the baseline model, the Word2Vec embeddings, and the BERT embeddings.

model	*accuracy*	$precision_w$	$recall_w$	$f1_w$	$f1_{mac}$
baseline	0.67	0.67	0.67	0.67	0.58
word2vec embeddings	0.47	0.43	0.47	0.44	0.26
bert embeddings	0.41	0.17	0.41	0.24	0.12

Table 7.9: Impact of Embedding Techniques on Model Performance

Surprisingly, the baseline is the clear winner of this experiment 7.9. This might stem from using embeddings in the baseline, which are only derived by the training data, and as such allowing for the best representation of the specific data.

7.4.3 Dense Layer

Additional dense layers using a non-linear activation function introduce variance to the model, potentially improving its performance. For this experiment, one additional dense layer with 64 units and the ReLU activation function was used.

Baseline - Dense Layer Comparison

The results in Table 7.10 show the performance comparison between the baseline model and the model with the additional dense layer.

model	*accuracy*	$precision_w$	$recall_w$	$f1_w$	$f1_{mac}$
baseline	0.67	0.67	0.67	0.67	0.58
dense	0.69	0.69	0.69	0.69	0.59

Table 7.10: Evaluation of Dense Layer Impact on Model Performance

The inclusion of an additional dense layer with 64 units and the ReLU activation function leads to improvements in the model's performance. The accuracy increased from 0.67 to 0.69, and similar increments were observed in precision, recall, and F1 scores, all moving from 0.67 to 0.69. Additionally, the macro F1 score improved from 0.58 to 0.59. These results suggest that the additional dense layer helps the model learn more complex representations of the input data by introducing non-linearity. This enhanced representation capability allows the model to capture more intricate patterns within the data, leading to better overall performance.

7.4.4 Convolutional Layer

For this experiment, a one-dimensional convolutional layer is used, which convolutes the input over a single temporal dimension followed by a max pooling layer (Jang et al., 2020). The layer was configured with 128 filters and a kernel size of 10.

Baseline - Additional Convolutional Layer Comparison

The results in Table 7.11 show the performance comparison between the baseline model and the model with the additional convolutional layer.

model	*accuracy*	$precision_w$	$recall_w$	$f1_w$	$f1_{mac}$
baseline	0.67	0.67	0.67	0.67	0.58
convolution layer	0.74	0.74	0.74	0.74	0.66

Table 7.11: Performance Comparison with the Addition of a Convolutional Layer

The results in table 7.11 reveals that the addition of a convolutional layer to the baseline model significantly enhances its performance. The accuracy improved from 0.67 to 0.74, and similar improvements were observed in precision, recall, and F1 scores, all increasing from 0.67 to 0.74. Additionally, the macro F1 score rose from 0.58 to 0.66. These results suggest that the convolutional layer's ability to focus on specific temporal features within the input data helps the model to generalize better and reduces the impact of noise and irrelevant features. The balanced improvement across both weighted and macro F1 scores indicates that the model's robustness has been enhanced, leading to better performance across different classes.

7.4.5 Attention Mechanism

The attention mechanism introduces self-attention after the LSTM layer, where each time step is compared and scored relative to all other time steps, and then passed to the classification softmax layer (Y. Wang et al., 2016).

Baseline - Attention Mechanism Comparison

The results in Table 7.12 show the performance comparison between the baseline model and the model using the additional attention mechanism.

model	*accuracy*	$precision_w$	$recall_w$	$f1_w$	$f1_{mac}$
baseline	0.67	0.67	0.67	0.67	0.58
attention	0.68	0.68	0.68	0.68	0.61

Table 7.12: Analysis of Attention Mechanism Impact on Model Performance

The inclusion of the attention mechanism leads to noticeable improvements in the model's performance. The accuracy increased slightly from 0.67 to 0.68, and similar small increments were observed in precision, recall, and F1 scores, all moving from 0.67 to 0.68. Additionally, the macro F1 score improved from 0.58 to 0.61. These results indicate that the attention mechanism helps the model focus on more relevant features by weighing the importance of each time step, which enhances the overall understanding of temporal dependencies in the data. Despite the relatively modest improvements in metrics, the attention mechanism contributes to a more nuanced and detailed representation of the input sequences, ultimately leading to better generalization and robustness.

7.5 Final Model

The final model was developed through a comprehensive tuning process to identify the optimal configuration. Various configurations were tested, including different combinations of convolutional layers, batch normalization, attention mechanisms, and dense layers. The tuning process involved:

- Utilizing a tuner to find the optimal model configuration.
- Experimenting with different optimizations, such as convolutional layers, batch normalization, and attention mechanisms.
- Running the models for up to 30 epochs to ensure optimal performance.
- Testing various layer quantities and configurations.

The selected final model is shown in figure 7.5 and the configuration includes:

- A Conv1D layer with 512 filters and a kernel size of 50.
- Max Pooling with a pool size of 25
- LSTM layer with 718 units.
- A Dense layer with 128 units.
- Batch size of 96.

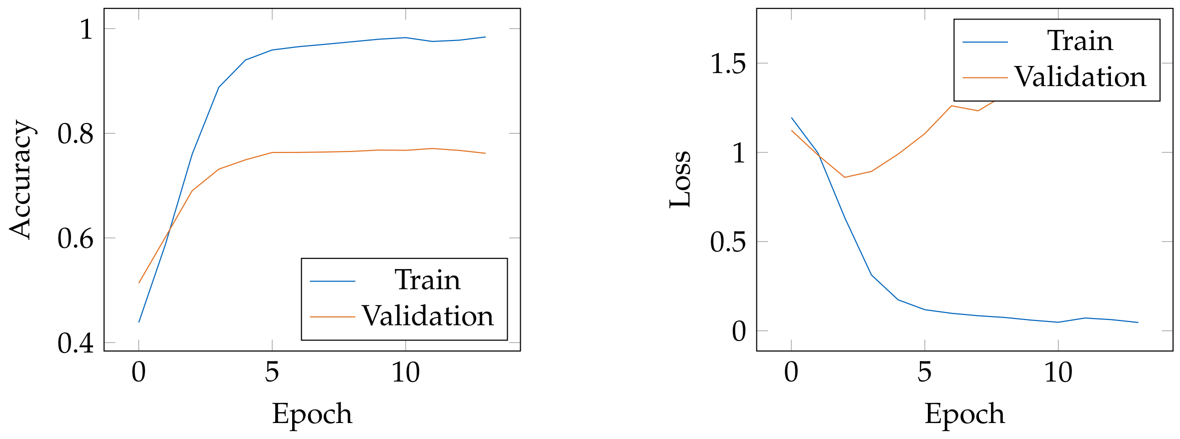
The results in Table 7.13 show the performance comparison between the baseline model and the final model and the final model using batch normalization.

model	$accuracy$	$precision_w$	$recall_w$	$f1_w$	$f1_{mac}$
baseline	0.67	0.67	0.67	0.67	0.58
final	0.78	0.78	0.78	0.78	0.74
final with batchnormalization	0.76	0.78	0.76	0.75	0.72

Table 7.13: Final Model Performance Comparison and Effect of Batch Normalization

The results in table 7.13 show that the final model, after optimization, achieved an accuracy of 0.78 and a macro F1 score of 0.74. These results indicate a strong overall performance, reflecting the model's ability to generalize well across different classes.

When batch normalization was applied to this final model, there was a slight decline in both key metrics. The accuracy decreased marginally to 0.76, and the macro F1 score dropped to 0.72. This suggests that the addition of batch normalization, while often beneficial in other contexts, did not contribute positively in this specific configuration. Instead, it may have introduced minor instability or failed to align effectively with the other components of the model architecture.

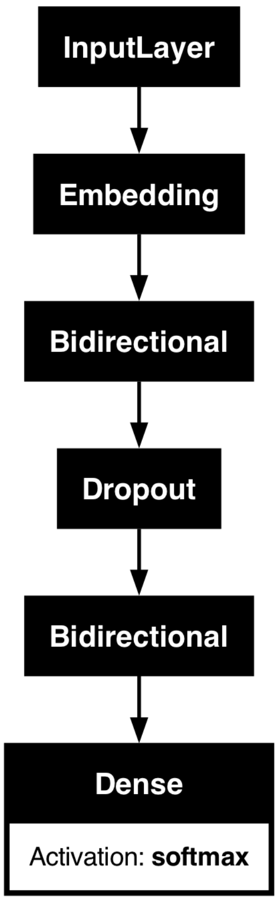


(a) Model Accuracy (b) Model Loss

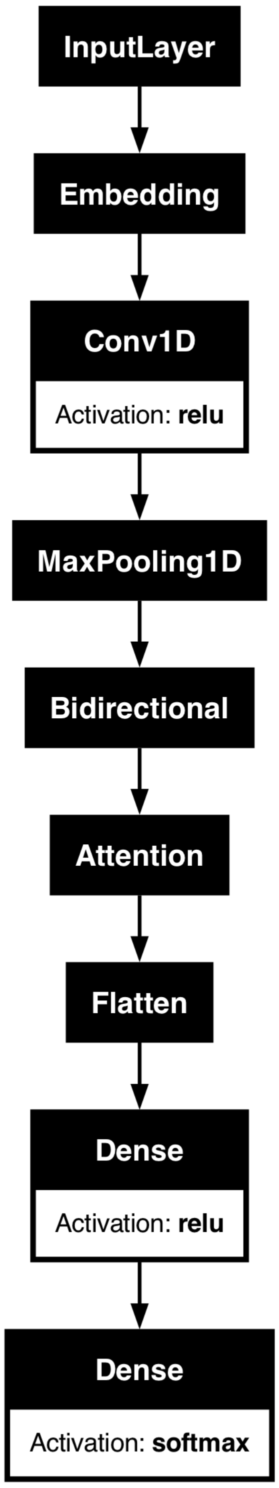
Figure 7.4: Final Model Accuracy and Loss during training

In the Model Accuracy plot 7.4, training accuracy quickly increases and stabilizes near 0.98, while validation accuracy levels off around 0.78. The noticeable gap between these curves indicates overfitting, where the model performs well on training data but struggles with unseen data.

In the Model Loss plot 7.4, training loss drops rapidly, nearing zero, while validation loss decreases initially but then slightly rises and stabilizes. This rise suggests the model is making incorrect predictions with high certainty, contributing to higher loss. Despite these issues, the model still achieves strong overall performance, though its high confidence in wrong predictions highlights the need for further tuning.



(a) Baseline LSTM



(b) Final LSTM

Figure 7.5: Comparison of final model structure to baseline model structure

8 LLM Implementation - AT

This chapter describes the implementation of the LLM Risk Predictor. Section 8.1 compares how well the representatives from the Encoder, Decoder and Encoder-Decoder LLM types predict injury risk. Then, the best-performing model[1] is selected as the baseline in Section 8.2. The baseline models are evaluated against variations of themselves, which are changed structurally and optimized for better performance in Section 8.3. The dataset on which these models are trained is defined in the chapter 4. Finally, one version of the LLM is chosen as the final risk prediction solution in section 8.4. For a fair comparison, only the smallest versions of the initial models, meaning with the smallest amount of parameters, is considered.

8.1 LLM Structure Selection

As already mentioned, LLMs have different structures based on how they employ a Transformer's Encoder and Decoder, which influences how LLMs perform on a certain task. Therefore, in this section, Encoder-only, Decoder-only, and Encoder-Decoder LLMs are first theoretically compared based on their structure and are then practically tested on their performance in predicting the urgency of injuries. Based on the results from the comparison and tests, models are recommended as the final solution. Additionally, factoring the model's strengths, to achieve an optimal result a prediction strategy would also be suggested, for example whether an LLM should classify or generate the results.

8.1.1 Encoder LLMs

Encoder LLMs utilize a stack of multiple Transformer Encoder layers (8.1), which are used to embed the input and additionally their context, and thereby excel at understanding. This structure is why Encoder LLMs need additional more fine-tuning or structures to generate text than for classification and NER tasks. The models with that structure that are going to be analyzed in this section are the popular BERT (Bidirectional Encoder Representations from Transformers) and its variants DistilBERT (Distilled version of BERT) and RoBERTa (Robustly Optimized BERT). (Devlin, 2018)(Y. Liu et al., 2019)(Sanh et al., 2019)

BERT

BERT is an LLM, developed by the Google AI Language team. Its most prominent feature is its bidirectional understanding, which was achieved by using bidirectional self-attention

[1]In this chapter, the word "model" refers to LLMs (Large Language Models)

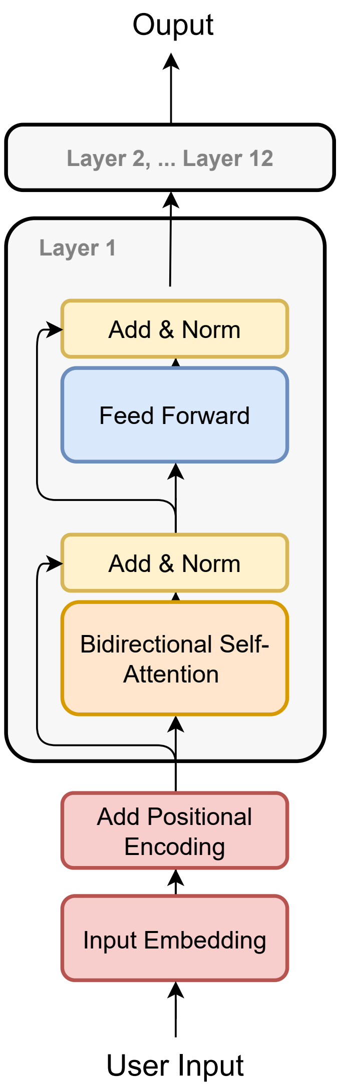


Figure 8.1: Layer structure of BERT, representative of Encoder LLMs (based on Devlin, 2018)

layers and a "masked language model" (MLM) pre-training objective. While Transformers encode the representation of a given token by conditionally factoring in the previous tokens (meaning left-to-right), BERT additionally considers the following tokens, which allow it to understand the full context of a token (from both the left and right side). However, given this left-to-right and right-to-left conditioning, if the model has access to the whole text sequence, including the token it has to predict, it might resort to directly using the target token instead of predicting it. Thereby, the model utilizes the MLM masking technique at data preparation time by masking 15% of the input tokens with a [MASK] token, or a random token or are left unchanged, which prevents the model from depending on any specific masking strategies. For generating the embeddings, BERT uses a WordPiece tokenizer. (Devlin, 2018)

From an implementational standpoint, two important factors to consider when choosing a specific BERT or BERT-derived model are whether it is case-sensitive and whether it supports one or multiple languages.

A *cased* model is case-sensitive, while an *uncased* model is case-insensitive. Because the data is in German, which differentiates nouns with a capital letter, using an uncased model might cause that it does not capture the linguistic properties correctly and, therefore, only cased version will be examined.
A monolingual BERT, such as German BERT, is trained solely on German texts, whereas a multiligual BERT is conditioned with a text corpus in multiple languages. According to Aßenmacher (Aßenmacher et al., 2021), German BERT models outperform their multilingual counterparts on a variety of tasks, such as relevance classification, sentiment classification, and aspect-based sentiment analysis within tweets. Sentiment classification is a type of multi-class text classification focused on identifying the sentiment expressed in text and we therefore assume that monolingual LLMs would perform better on classification tasks using German medical data.

Therefore, in the upcoming sections and subsections, only cased German LLMs will be considered. Regarding BERT, the exact chosen LLM for the implementation comparison is *BERT base cased German* because it fulfills all of the mentioned requirements.[2] Textbox 8.2 shows how *BERT base cased German* fails to diagnose a patient from our study dataset, because by design it uses masking to predict only the next token. However, Section 8.2.1 corrects this inability of BERT.

RoBERTa

RoBERTa is a version of BERT that was trained on 10 times more textual data. Additionally, the masking technique is dynamic and is performed during training instead of during pre-processing, hence the masking configurations are repeated fewer times and are more efficacious. (Y. Liu et al., 2019) The original RoBERTa, however, was trained on English

[2]https://huggingface.co/google-bert/bert-base-german-cased

German BERT Classification through masking

USER: X years old male X years ago anterior cruciate ligament rupture while playing soccer. In hospital arthroscopy with stump resection. In 20XX Giving way while playing soccer. Patient wants to be able to jog and do winter sports. My right knee has a discrete to moderate swelling, and there is no hyperextension and no extension deficit. Flexion of the knee is quite possible and end-grade. There is pain on the medial side of the knee joint when pressing. Here are 5 classification labels that describe the injured knee structure and the severity of the injury: *meniscus-urgent, meniscus-minor, cruciate ligament-urgent, cruciate-ligament-minor, other-minor*. Of these, the most appropriate for the symptom description is the label [MASK].

BERT base cased German: ##name

Figure 8.2: Test of basic BERT classficiation skills

only. In order to employ it for German data, the XLM-RoBERTa model version has to be used. (Conneau et al., 2019) XML is multi-lingual and was trained on CommonCrawl data in 100 different languages using specific cross-lingual language models (XLMs) techniques (Lample & Conneau, 2019). Because XLM-RoBERTa ouperforms both BERT and RoBERTa for language understanding, the XLM-RoBERTa specialized with Fast Overlapping Token Combinations Using Sparsemax (FOCUS) method in German[3] will be considered as a BERT solution optimization. (Dobler & de Melo, 2023)

DistilBERT

DistilBERT has 40% of the size of BERT and 97% of its performance in language understanding (Sanh et al., 2019). This model was developed to be lighter and hence less time- and compute-intensive while still performing well. However, in this thesis it is not restricted how fast the Risk Predictor computes its results, but the model understanding of the data is highly prioritized. Therefore, as BERT and RoBERTa both perform in a timely manner too, DistilBERT was not considered further.

8.1.2 Decoder LLMs

Decoder LLMs employ a Decoder-only stack and as they do not have an Encoder, they perform input encoding in the Decoder stack. This encoding, however, is different than the one performed by Encoder LLMs, because they are unidirectional and have a different training objective. These models process output left-to-right, which means that they have only half of the context for a given token and their understanding is not as comprehensive as that of an

[3]https://huggingface.co/konstantindobler/xlm-roberta-base-focus-german

Encoder model. Decoder LLMs are also commonly trained using a Causal Language Modeling (CLM) method that makes the models predict the next token in a sequence based on the tokens before it. This next-token prediction strategy is characterized as auto-regressive. For these reasons, these LLMs are proficient in generating text, but not inherently in classification, which is why they are commonly referred to as GenAI models. From the Decoder family, GPT (Generative Pre-training Transformer) and Llama (Large Language Model Meta AI) will be examined as potential Risk Predictors. (Radford et al., 2019b) (Touvron et al., 2023)

GPT

GPT is a Decoder LLM, developed by OpenAI. Currently four generations of the model (GPT 1-4) have been developed. Each they address the issues of its predecessors by training it on more data among others techniques. GPT-4 is multi-modal, which means that it accepts several non-textual input types, like images. It also aims to be more factually correct, hallucinate less and be less biased with the help of post-training techniques, such as RLHF (Reinforcement learning from human feedback). RLHF is based on a reward model, which judges how close the LLM outputs are to human preferences and gives it a score during an internal fine-tuning. The model incorporates the score and recalibrates itself to better its results. Due to these improvements, GPT-4 manages to outperform many models based on academic and professional benchmarks. (Achiam et al., 2023) However, GPT-1 and -2 are outperformed by many newer and open-source models and the bigger problem is that GPT-3 and -4 are proprietary and only available through an API. This API-only access means that the data would be communicated to an external server and company employees would have access to it under specific circumstances[4]. Thereby, to ensure full data privacy, GPT would not be considered as a solutions in this thesis.

Llama

Llama is a Decoder LLM, developed by the AI @ Meta team (Team & Meta, 2024). Despite its compact size, ranging from 8 to 405B parameters, the newest Llama 3.1 outperforms GPT-4, which is significantly bigger (1760B parameters). Although Llama also consists of a stack of Decoders, their structure differs significantly from those of a Transformer or other Decoder LLMs. Instead of capturing the absolute positions of tokens using positional embeddings, Llama monitors position information with Rotary Positional Embeddings (RoPE) (Su et al., 2021), which encodes the relative positions dependencies with a rotation matrix. Additionally, instead of two multi-head attention sublayers, it has one Grouped Query Attention (GQA) sublayer with masking that processes the queries Q into smaller groups, which are processed similarly. Consequently, this attention mechanism has to compute uniquely fewer components than rival LLMs, which makes it use less memory and compute faster. In this process it also uses Root Mean Square (RMS) normalization, which computes root mean square of the input features and not the mean and variance unlike Layer normalization, and a SwiGLU activation function instead of ReLU for its Feed Forward sublayer. Lastly, similarly to GPT it employs

[4]https://openai.com/enterprise-privacy/

German Llama 3 Instruct Text Generation

SYSTEM PROMPT: You are a doctor who receives a description of the symptoms of a knee problem from a patient. You must diagnose the patient and predict that diagnosis. Give only the final diagnosis.

USER: X years old male X years ago anterior cruciate ligament rupture while playing soccer. Arthroscopy with stump resection in Bergneustadt. In December 2013 Giving way while playing soccer. Patient wants to be able to go jogging and do winter sports. My right knee has discrete to moderate swelling and there is no hyperextension or extension deficit. Flexion of the knee is well possible and full range of motion. There is pain on the medial side of the knee joint when pressing. The diagnosis is:

Llama 3 Instruct 8B: Posttraumatic osteoarthritis after anterior cruciate ligament rupture

Figure 8.3: Basic German Llama 3 Instruct diagnosis through text generation

RLHF to post-train. Llama's improved features and good performance make it a suitable risk prediction solution.

In this thesis we further examine Llama 3 Instruct 8B [5], which was fine-tuned to understand German texts and instruction-tuned to respond in a dialogue or chat manner. Textbox 8.3 shows a basic example of German Llama 3 Instruct diagnosing a patient from our study dataset based on their anamnesis and interpreted examination.

[5]https://huggingface.co/DiscoResearch/Llama3-DiscoLeo-Instruct-8B-v0.1

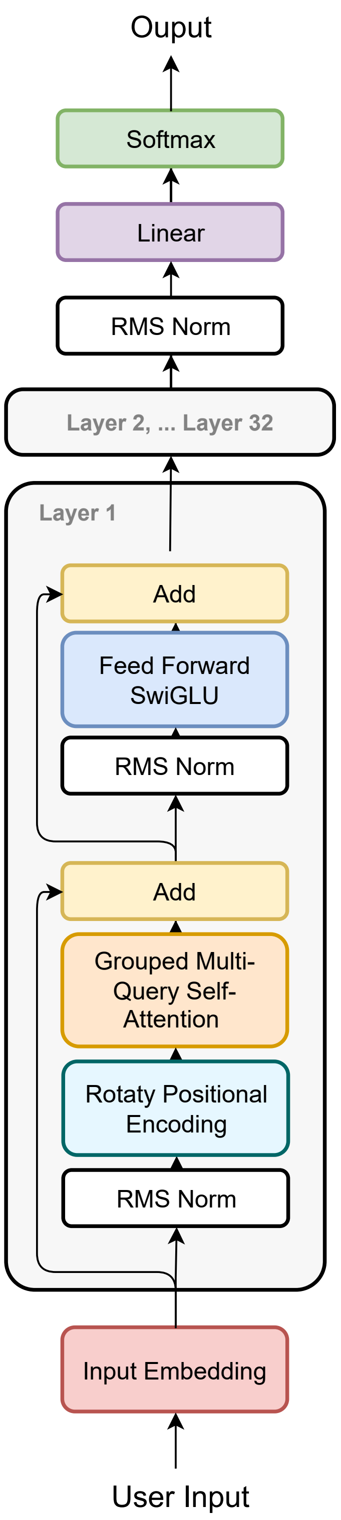


Figure 8.4: Layer structure of the Llama Model, representative of Decoder LLMs (based on Team and Meta, 2024)

8.1.3 Encoder-Decoder LLMs

Encoder-Decoder LLMs are similar to the original Transformers in terms of structure. They have one Encoder and one Decoder with varying amounts of layers, which makes them excel at generating output that depends on the entire input sequence, such as translations and text summaries. The most popular representative is T5 (Text-to-Text Transfer Transformer). T5 employs the general Transformer structure with small changes and ranges from 6 to 24 sublayers per Encoder and Decoder. Instead of the Transformer absolute position encoding, it encodes the relative positions to generalize better over inputs of different length. T5 also uses a singular embeddings layer that spans both over the Encoder and Decoder to minimizes the parameters and creates an output more cohesive with the input. Another difference from the Transformers is that it performs layer normalization separately from the additive processes, before each sublayer. T5 was also trained with a Span-Level Masked Language Modeling (MLM) objective, which is similar to the MLM technique of BERT, but masks continues token sequeunces and not singular tokens, and it uses the SentencePiece tokenizer unlike BERT's WordPiece. Finally, when T5 is used for downstream tasks (translation, sentence similarity etc.), the user prompt should have this task as a prefix, for example: "Translate from English to German: (...)". Because T5's strengths do not fit with the thesis objective, it will not be investigated further.

8.2 Establish Baseline Model

As already discussed in Section 8.1 LLM Structure Selection, both BERT and Llama need additional structural features or optimization to properly predict patient risk. To improve the quality of results, we will leverage the strengths of each model by examining BERT for classification tasks and Llama for inference purposes. In this section, we bring the models to a state, in which they can respectively classify injury risk or generate patient diagnosis with urgency level, through minimal changes and use these as a baseline for any further experiments.

8.2.1 Classification

As seen in Figure 8.2, BERT cannot innately assign a label to a text. To overcome this limitation, BERT has been tested in a TFBertForSequenceClassification configuration with the accompanying tokenizer BertTokenizer. TFBertForSequenceClassification[6] is a TensorFlow-optimized wrapper from the transformers Python package that adds a sequence classification/regression head (a linear layer on top of the pooled output) to the original model. TFBertForMultipleChoice[7] is another wrapper that performs multiple choice classification, however, unlike TFBertForSequenceClassification, which assigns each data entry a label from a static set (the classification goal of the risk prediction solution), it learns to choose from labels that vary

[6]https://huggingface.co/transformers/v2.10.0/_modules/transformers/modeling_tf_bert.html#TFBertForSequenceClassification

[7]https://huggingface.co/transformers/v2.10.0/model_doc/bert.html#tfbertformultiplechoice

across samples. This difference in task definition of the two wrappers is why TFBertForSequenceClassification was preferred. Therefore, as the baseline configuration we chose BERT with a classification layer (TFBertForSequenceClassification) and trained for 3 epochs on the study data (described in Section 4.1) using the default training hyperparameters.

We evaluated the metrics accuracy, precision, recall, F1 macro score, and F1 weighted score. Among these, the F1 macro score was the most significant as it best represents the classification performance on imbalanced datasets like ours. This baseline setup performed on our default test dataset as follows, according to Table 8.1: The F1 macro score (35%) is quite low compared to the accuracy (56%), which means that the model has problems classifying correctly the minority class cruciate-ligament-minor. This flaw is also visualized in the confusion matrix in Figure 8.5, because the model did not predict the label *cruciate ligament minor* for any sample. Section 8.3 targets this flaw and strives to improve the overall performance.

model	$accuracy$	$precision_w$	$recall_w$	$f1_w$	$f1_{mac}$
bert-baseline	0.56	0.58	0.56	0.53	0.35

Table 8.1: Evaluation of BERT Baseline Model

8.2.2 Free text generation

Contrary to BERT, Llama is able to generate text the desired output, detailed diagnosis and risk level, relatively well. The Llama baseline utilizes German fine-tuned Llama 3 Instruct 8B with a chat text generation structure. This baseline was tested on five samples using the system prompt and prompt in Figure 8.3. Each of the samples tested one of the classification label categories (meniscus urgent/minor, cruciate ligament urgent/minor and other minor injuries). The test results in Table 8.2 show that the responses are not uniform, as they vary in structure. Additionally, the model's accuracy cannot be properly evaluated without a medical specialist, because diagnoses like "Other meniscus damage: other and unspecified part of the medial meniscus" are not fully conclusive. If the model predicts a correct diagnosis but phrases it differently in a more concrete manner, it may still be counted as inaccurate, because metrics like cosine similarity rely solely on linguistic properties and fail to capture medical accuracy. Thereby, Llama was not considered for the performance optimization techniques, described in the next Section 8.3, but was further discussed in Chapter 10.2 Future work.

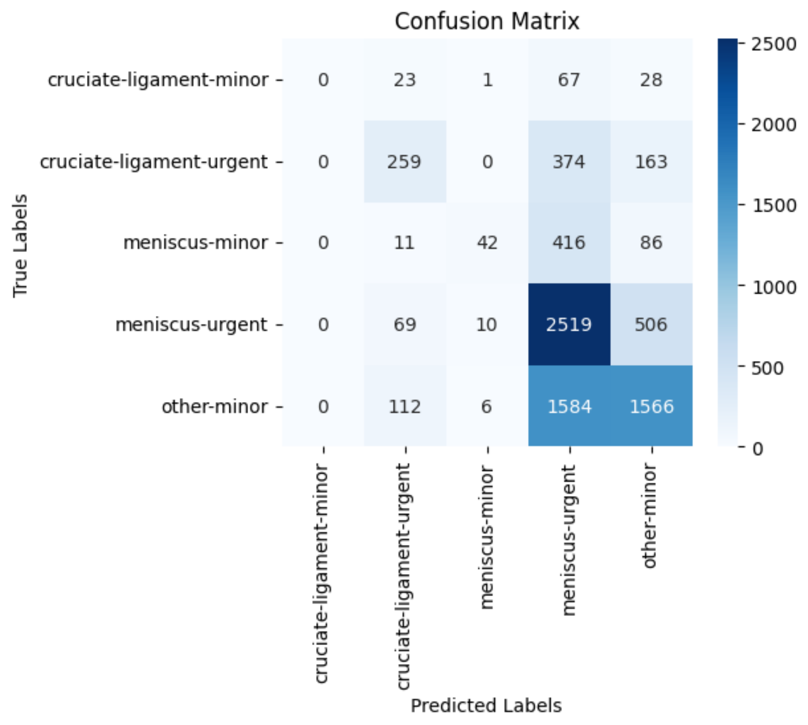


Figure 8.5: BERT baseline confusion matrix

Doctor diagnosis	**Llama 3 Instruct diagnosis**
Chronic instability of the knee joint: anterior cruciate ligament	The diagnosis is: Follow-up after anterior cruciate ligament rupture with residual complaints. The urgency level is classified as mild.
Meniscus damage due to old tear or old injury: other and unspecified part of the medial meniscus	The final diagnosis is: Gonarthrosis. The urgency level is mild.
Other spontaneous rupture of the medial collateral ligament of the knee joint [Lig. collaterale tibiale]	The final diagnosis is: Patellar tip syndrome (Jumper's Knee). The urgency level is mild.
Other meniscus damage: other and unspecified part of the medial meniscus	The diagnosis is: Patellofemoral pain syndrome (Runner's knee). The urgency level is mild.
Other internal derangements of the knee joint: unspecified ligament	The diagnosis is: Patellofemoral pain syndrome. This is a mild condition.

Table 8.2: Comparison of doctor diagnosis and Llama 3 Instruct diagnosis

8.3 Performance optimization techniques

In this section, we try to optimize the BERT classifier using different methods, whose performance we compare with the baseline. Each comparison is supposed to showcase which technique the final risk predictor should employ. The optimizations are split in hyperparameter-related, structural, training and fine-tuning, based on the way they are enhancing the model.

8.3.1 Hyperparameter optimization

Hyperparameters are the configurations that determine how a model trains, from which this section analyzes the amount of epochs, the learning rate and the batch size and aims to find values for them that yield the best performance.

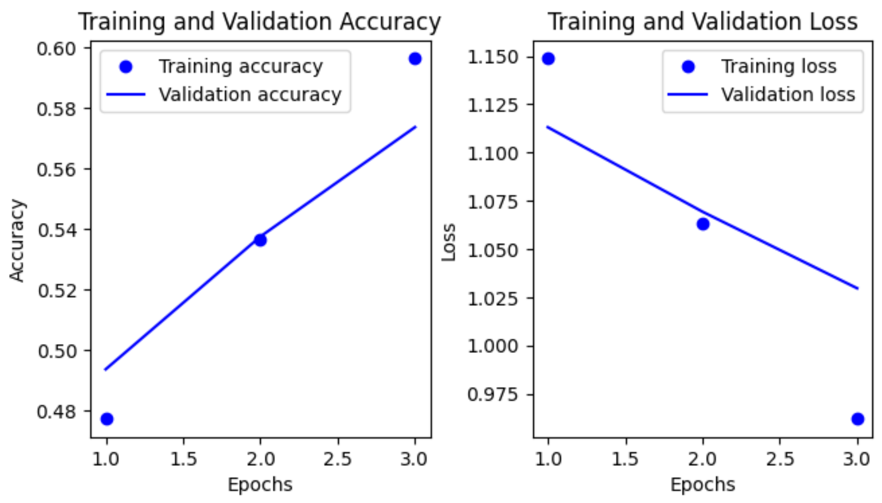
Epochs

Each epoch denotes one passing of the data through the model while training. Models that are trained from scratch, such as LSTMs, require a higher amount of epochs than pre-trained models like BERT. Pretrained models are actually prone to overfitting or even catastrophic forgetting with similar amounts. Moreover, our study dataset, which contains roughly 12 million words[8] is rather small for BERT that was trained on 3,3 billion words, which usually causes overfitting. The BERT development team recommends for further training/fine-tuning using between 2 and 4 epochs, which is the reason for using 3 epochs for the baseline (Devlin, 2018). However, the baseline results were not encouraging, so we also explored training using 10 epochs.

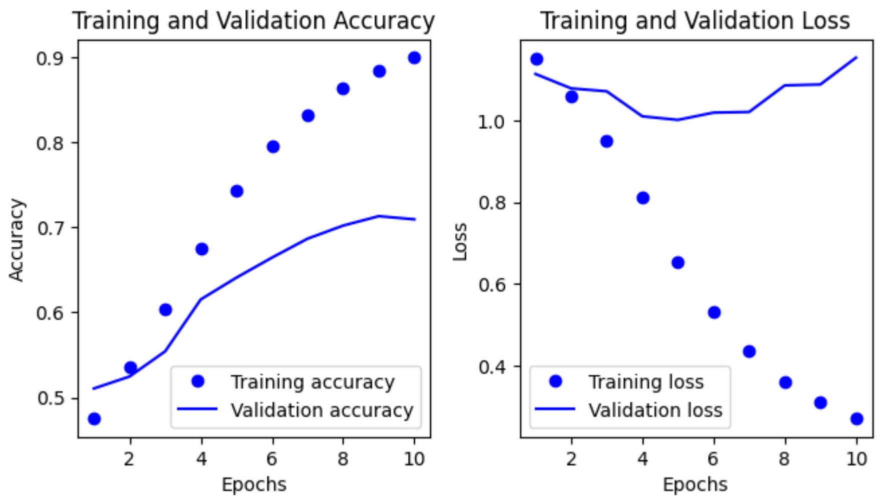
Table 8.3 compares the two BERT versions - one trained for 3 epochs ("bert-baseline") and the other for 10 epochs ("bert-epochs") - and shows that the model performs better when it trains for more epochs. Specifically, the accuracy increases from 56% in the baseline model to 72% in the 10-epoch model, which indicates that extending training allows the model predict more correctly. More notably, the F1 macro score, which balances precision and recall across all classes, jumps from 35% in the baseline to 65% for the extended training model. This increase suggests that the "bert-epochs" model not only is more accurate but also handles class imbalances, making it more reliable for predicting across different classes.

Figure 8.6 displays that the model trained for 10 epochs begins to overfit as it begins to more accurately predict the training data but less effectively the validation data. While the model specializes highly in predicting the training data, it struggles to generalize new, unseen data based on the 10-20% difference in training and validation accuracy in Figure 8.6b, however, the accuracy still slowly improves despite the plateauing and increasing of the validation loss.

[8]42,000 samples with maximum of 400 tokens each and 1 token equals 0.75 words according to https://help.openai.com/en/articles/4936856-what-are-tokens-and-how-to-count-them



(a) LLM Baseline (3 epochs)



(b) LLM 10 Epochs

Figure 8.6: Comparison of model training using 3 epochs (baseline) vs. 10 epochs

model	*accuracy*	*precision*$_w$	*recall*$_w$	$f1_w$	$f1_{mac}$
bert-baseline	0.56	0.58	0.56	0.53	0.35
bert-epochs	0.72	0.73	0.72	0.72	0.65

Table 8.3: Comparison of BERT Performance with 3 (BERT-Baseline) vs. 10 (BERT-Epochs) Training Epochs

Learning Rate

The speed at which a model learns the training data (learning rate) is vital for preventing the model from erasing all of its pre-trained knowledge and substituting it with new knowledge (catastrophic forgetting). Rates, such as 4e-4 (denoted in mathematics as 4.10^{-4}), are too aggressive and do not allow the model to converge (Sun et al., 2019). For that reason, the learning rates 2e-5, 3e-5 (baseline) and 4e-5, which the original BERT paper also utilized, were examined. (Devlin, 2018).

model	*accuracy*	$precision_w$	$recall_w$	$f1_w$	$f1_{mac}$
bert-baseline	0.56	0.58	0.56	0.53	0.35
bert-2e-5-learning-rate	0.57	0.57	0.57	0.55	0.35
bert-4e-5-learning-rate	0.59	0.59	0.59	0.57	0.39

Table 8.4: Impact of Different Learning Rates on BERT Model Performance

Table 8.4 presents the results from comparing BERT training using the learning rates 2e-5, 3e-5 (baseline) and 4e-5. These results indicate that a model with a slightly higher learning rate (4e-5) performs 1-4% better than those with 2e-5 and 3e-5 across all metrics, potentially because the model converges faster and generalizes better. It appears that the 4e-5 learning rate stays comfortably below the catastrophic forgetting threshold, as the table shows no signs of this issue, only a modest improvement in performance.

Batch size

Batch size is the size of the chunks in which the model "sees" the data before updating its parameters while training. Batch size is typically chosen as powers of two (e.g., 32, 64, 128) because this optimizes memory usage and computational efficiency on most hardware. It determines how individually the model tailors its output to a certain datapoint. For example a batch size of 1 will make the model too focused on the singular sample and would create noise. Therefore, bigger batch calculate a more precise gradient with respect to the loss function, making the gradient less noisy. (Popel & Bojar, 2018) This test compares using a batch size of 64, in the current baseline, and 32, as per Devlin, 2018. The model also shuffles the batches independently for each epoch, ensuring that the batches uniquely composed. The model implements this batching by allocating a buffer of static size and randomly sampling it when creating the training data chunks, which helps the model remain unbiased, converge faster and be more accurate. (Zhong et al., 2023)

model	*accuracy*	$precision_w$	$recall_w$	$f1_w$	$f1_{mac}$
bert-baseline	0.56	0.58	0.56	0.53	0.35
bert-32-batch	0.58	0.59	0.58	0.57	0.38

Table 8.5: Comparison of BERT Performance with Batch Sizes of 64 (bertbBaseline) and 32 (bert-32-batch)

The table 8.5 compares the performance of BERT versions with two different batch sizes: 64 (baseline) and 32. The results suggest that a smaller batch size of 32, compared to the baseline of 64, performs better across all metrics. The modest improvements in accuracy, precision, recall, and F1 scores indicate that the model benefits from updating its weights in finer granularity that smaller batches provide. The improved performance also aligns with the idea that smaller batch sizes help the model generalize better by making the gradient updates less deterministic and thus allowing the model to explore a broader range of solutions. (Popel & Bojar, 2018)

8.3.2 Training

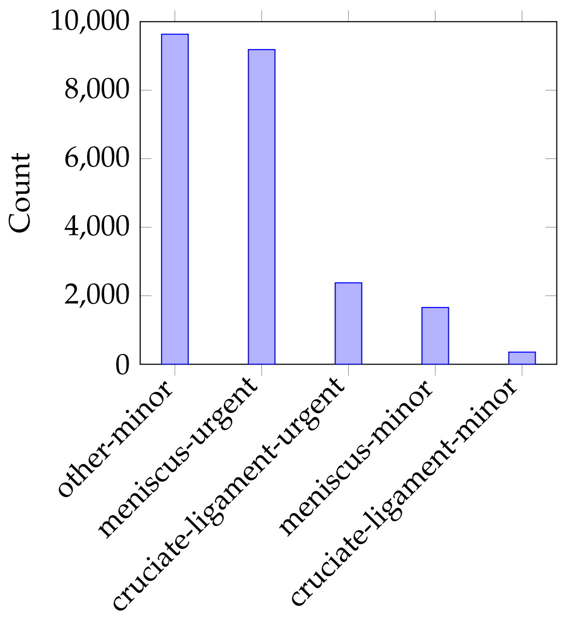
Dataset balance

The study dataset, outlined in chapter 4 Study Design, is used by the LLM for training and validation. This dataset, however, is imbalanced based on discrepancy between label amounts. The label distribution in Figure 8.7 showcases the big gap in the amount of meniscus, highlighting the class *meniscus-urgent* as a majority and *cruciate-ligament-minor* as a minority class. Such imbalance causes bias and higher inaccuracy while training, hence undersampling, oversampling and class weight are examined as balancing methodologies.

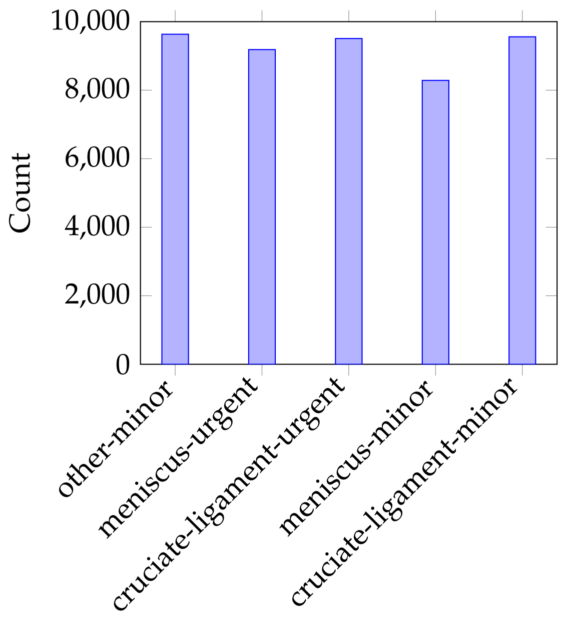
Undersampling reduces the size of the majority (more prominent) classes by selectively removing some of their data, thereby equalizing their size with the minority classes and achieving an even distribution. (Mohammed et al., 2020b) The method RandomUnderSampler[9] was researched for this purpose, as it undersamples the majority class(es) by randomly choosing samples and removing them, but it accepts input only in matrix representation, such as feature vectors or embeddings. This representation was impossible to achieve for our classification dataset while preserving BERT's integrity, because BERT internally computes the text embeddings. Therefore, the end undersampling solution was a manual algorithm that discards random data points from all classes except the smallest criciate-ligament-minor class until they reach the minority class' size, as shown on Figure 8.7.

Oversampling duplicates entries or generates synthetic data to increase the size of the minority classes in order to create a more balanced dataset.(Mohammed et al., 2020b) One

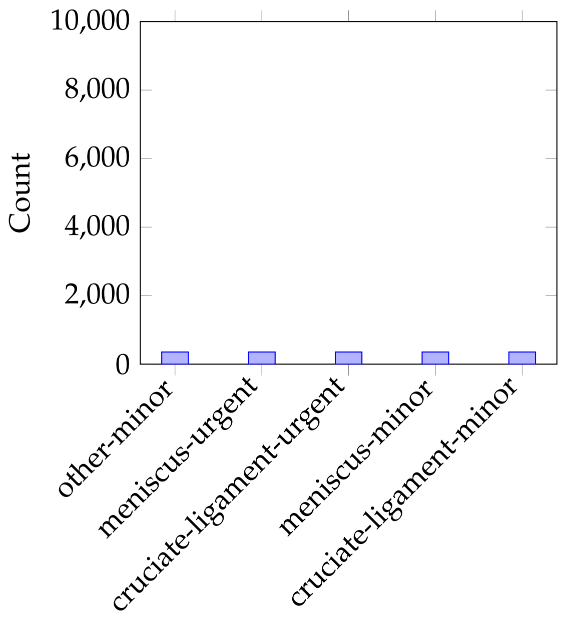
[9]https://imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.RandomUnderSampler.html



(a) Original Distribution



(b) Oversampled Distribution



(c) Undersampled Distribution

Figure 8.7: Distributions of Original, Oversampled, and Undersampled Study Dataset

of the most common approaches is SMOTE (Synthetic Minority Over-sampling Technique), which augments the original dataset using interpolation on the minority class (Chawla et al., 2002). For a certain fraction of minority data points, SMOTE calculates its k-nearest neighbors, selects one of them and interpolates between this neighbor and the minority class sample to synthetically create a new entry. However, SMOTE can only focus on one minority class and, similarly to the RandomUnderSampler, it requires feature vectors as input. Because of these drawbacks, instead of SMOTE, the oversampling was done with a manual that randomly samples all classes except for the three majority classes and duplicates these samples to achieve a more balanced distribution, displayed in Figure 8.7.

Class weights are another method to balance a dataset during training, but without actually altering it. Class weights represent how important a class is based on its frequency.(Shimodaira, 2000) The values are commonly computed as an inverse function of the class likelihood, meaning that the model pays more attention to the minority classes than the majority classes. This technique aims to prevent class bias to achieve a better performance across all classes. For this comparison, the class weight for a class *i* is computed as:

$$weight_i = \frac{\text{Number of samples}}{(\text{Number of classes} * \text{Occurrences of i}})$$

model	$accuracy$	$precision_w$	$recall_w$	$f1_w$	$f1_{mac}$
bert-baseline	0.56	0.58	0.56	0.53	0.35
bert-oversampled	0.58	0.63	0.58	0.57	0.57
bert-undersampled	0.31	0.45	0.31	0.34	0.26
bert-class-weights	0.43	0.56	0.43	0.47	0.37

Table 8.6: Comparison of BERT Results Using Baseline, Oversampling, Undersampling, and Class Weights

Table 8.6 showcases that oversampling is most effectively balances the dataset in this context, achieving the highest accuracy and F1 macro score (58% and 57%, respectively). The undersampled dataset causes the lowest performance across all metrics, more precisely the accuracy drops to 31% and the F1 macro score falls to 26%. This decline suggests that undersampling, which equalizes class distributions by reducing the size of the majority classes (as shown in Figure 8.7 (c)), leads to a loss of valuable information. The small sample size likely hinders the model's ability to generalize, resulting in poor performance. Additionally, the confusion matrix in Figure 8.8 that displays the class weight results suggests that the class weights help the model balance between classes, but most importantly prevents it from differentiating the labels, which makes class weights inferior to the oversampling technique.

RoBERTa

RoBERTa is a variant of BERT shares the same structure, but differs in training methodologies and knowledge base. This training difference is why, as already mentioned in Section 8.1.1,

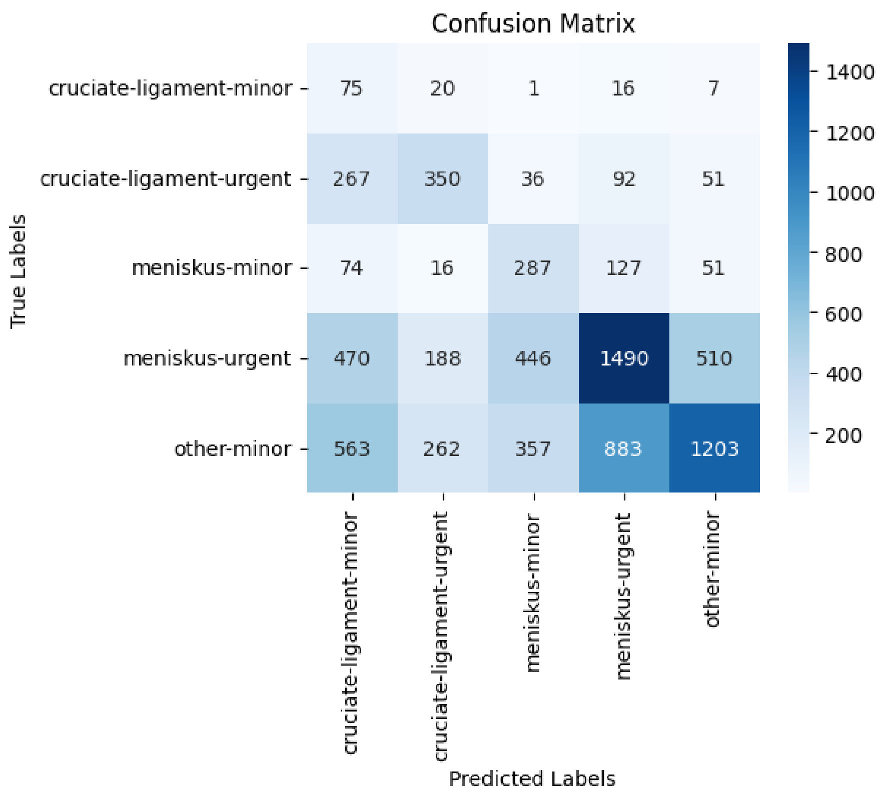


Figure 8.8: Results of BERT classification using class weights

XLM-RoBERTa FOCUS German model is examined more closely here. Like BERT, it also requires a wrapper, more precisely TFXLMRobertaForSequenceClassification, which utilizes a sequence classification/regression head (a linear layer after output pooling) along with the XLMRobertaTokenizer to classify text.

model	*accuracy*	$precision_w$	$recall_w$	$f1_w$	$f1_{mac}$
bert-baseline	0.56	0.58	0.56	0.53	0.35
roberta	0.1	0.01	0.1	0.02	0.04

Table 8.7: Comparison of BERT Baseline and RoBERTa Results

The comparison between the performance of the BERT baseline model and the RoBERTa model in Table 8.7 reveals a stark contrast in effectiveness. BERT, with an accuracy of 56%, significantly outperforms RoBERTa, which only achieves an accuracy of 1%. This dramatic difference extends across all evaluated metrics: BERT maintains a precision of 58%, recall of 56%, and an F1 macro score of 35%, whereas RoBERTa's precision, recall, and F1 macro score are all nearly negligible, sitting at 1%, 10%, and 4%, respectively. These results suggest that while RoBERTa is generally considered an improved variant of BERT due to its enhanced training methodologies and knowledge base, in this particular case, it performs far worse than the BERT baseline. This poor performance could be due to RoBERTa being unable to understand the dataset domain.

8.3.3 Fine-tuning for medical purposes

BERT as a general-purpose model does not specialize in domains like medicine. However, Bressem et al., 2023 introduced MedBERT.de, a fine-tuned version of BERT trained on 4.7 million German medical texts, including clinical notes, research papers, and more. This fine-tuning enhances BERT's ability to understand of medical jargon and context, making it more suitable for tasks, such as medical text classification and diagnosis prediction.

model	*accuracy*	$precision_w$	$recall_w$	$f1_w$	$f1_{mac}$
bert-baseline	0.56	0.58	0.56	0.53	0.35
bert-fine-tuning-med	0.58	0.57	0.58	0.56	0.39

Table 8.8: Comparison of BERT with and without Fine-Tuning for Medical Knowledge

Table 8.8 shows that fine-tuning on a large corpus of medical texts, as done with MedBERT.de, helps the model better understand the medical domain. While the improvements are modest, the increase in accuracy and F1 macro score (accordingly with 2% and 4%) demonstrates that MedBERT.de is more adept at handling the nuances of medical language and excels in specialized tasks, such as medical text classification and diagnosis prediction.

8.3.4 Structural optimizations

model	*accuracy*	$precision_w$	$recall_w$	$f1_w$	$f1_{mac}$
bert-baseline	0.56	0.58	0.56	0.53	0.35
bert-pooling-max	0.53	0.53	0.53	0.51	0.33
bert-pooling-average	0.55	0.58	0.55	0.52	0.37

Table 8.9: Comparison of BERT using max and using average pooling

This section discusses how extending BERT with a pooling layer and the necessary accompanying structural changes influence the risk predictor's performance. The extensions are inspired by the implementation of RuPool-BERT 265 (Blinov et al., 2020), which is a Russian BERT model trained to classify 256 ICD codes based on patient Electronic Health Records (EHRs). This LLM shares a few characteristics with the LLM risk predictor suggestion, namely both are monolingual in a non-English language and their objective is to primarily classify health conditions based on textual data. Due to these similarities the structure of RuPool-BERT 265 could benefit the LLM solution. Unlike RuPool-BERT 265, the LLM risk predictor would contain only a Global Max or Average Pooling layer (GlobalMaxPooling1D or GlobalAveragePooling1D) instead of a combination, but will still employ the Dropout and Dense Classification layer, which Figure 8.9 showcases. The Pooling layer reduces the dimensionality by globally aggregating the outputs across the entire sequence (which is 1D data), either by taking the maximum or the mean, as opposed to just using a local patch of the data (Lin et al., 2013). The Dropout layer creates a slightly different structure every epoch by setting 30% of the neural network nodes to 0, which regularizes and prevents overfitting, and the Dense layer is the fully-connected classification layer that outputs the label probabilities.

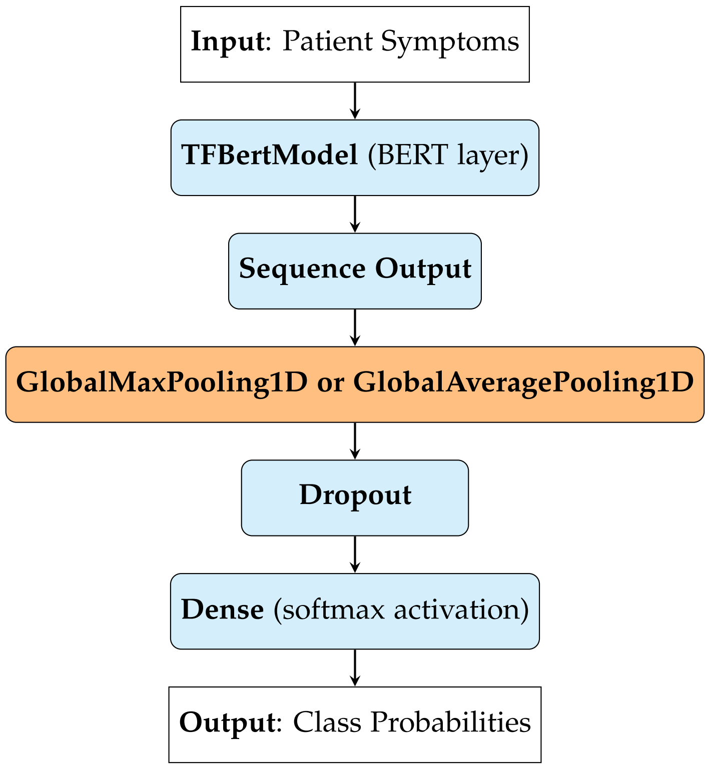

Input: Patient Symptoms
TFBertModel (BERT layer)
Sequence Output
GlobalMaxPooling1D or GlobalAveragePooling1D
Dropout
Dense (softmax activation)
Output: Class Probabilities

Figure 8.9: LLM Risk Predictor structure using a Pooling layer (in orange), together with a Dropout and Dense Classification layer

Table 8.9 compares the performance of BERT when using two different global pooling strategies: max pooling and average pooling, in comparison to the baseline BERT model. The results indicate that adding a global pooling layer (either max or average) together with Dropout and Dense layers on top of BERT (see Figure 8.9) slightly reduces the accuracy compared to the baseline model. Max pooling appears to be too selective, possibly overlooking important features that are not the most dominant in the sequence, leading to a noticeable drop in performance. On the other hand, average pooling, while still underperforming compared to the baseline, maintains a better balance by considering all features equally, resulting in a smaller drop in overall performance. The increased F1 macro score of average pooling, however, suggests that it most likely prevents overfitting and needs more training epochs to converge to a better result.

8.4 Final model

The final LLM Risk Predictor incorporates all the optimizations from the previous sections identified to enhance BERT, with a particular focus on improving the F1 macro score. This model was trained for 10 epochs, using a batch size of 32 to refine weight adjustments, and a learning rate to 4e-5 for a balanced approach to learning speed and stability. Additionally, oversampling of the dataset addresses class imbalance, and the fine-tuning for medical helps the model better understand domain-specific language. Finally, an average pooling layer aggregates the information evenly across sequences, contributing to the model's overall robustness and effectiveness.

model	$accuracy$	$precision_w$	$recall_w$	$f1_w$	$f1_{mac}$
bert-baseline	0.56	0.58	0.56	0.53	0.35
bert-final	0.74	0.75	0.74	0.74	0.7

Table 8.10: Comparison of BERT baseline model and final BERT Risk Predictor

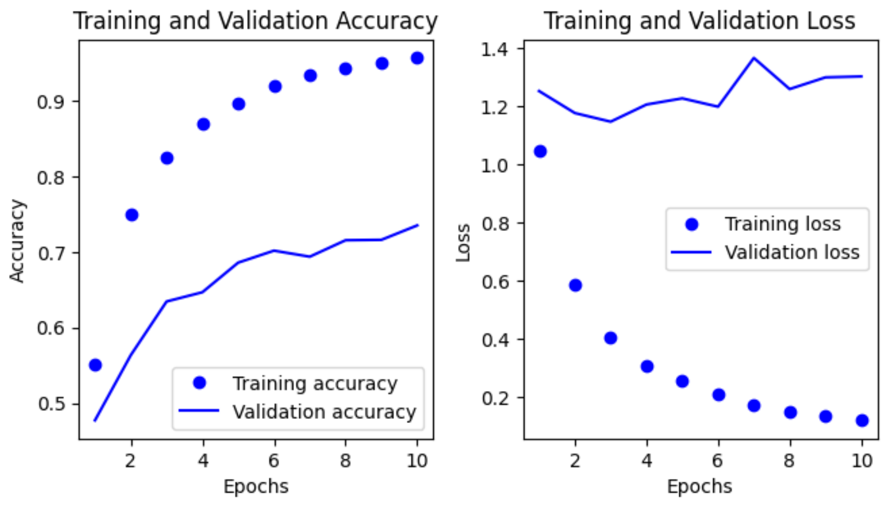


Figure 8.10: Accuracy and loss during training of final LLM Risk Predictor

Comparing the final model with the baseline in Table 8.10 confirms that the optimization indeed improve BERT's performance all-round and enhance both the accuracy (to 74%) and robustness of the model in the medical injury risk prediction task. Although Figure 8.10 indicates that BERT overfits, this overfitting does not impair the accuracy or F1 macro score.

9 Evaluation LSTM vs. LLM - AT

This chapter evaluates the LSTM and LLM risk prediction algorithms, implemented in this thesis. Section 9.1 compares the two algorithm configurations trained on two version of the original dataset - one with interpreted examinations and the other split into keywords using RAKE. We then evaluate how well the best-performing algorithm and dataset combination can distinguish knee injuries regardless of severity and prioritize patients effectively. Section 9.2 outlines threats to the validity of our research by analyzing our results on the unaltered anamnesis and examination data.

9.1 Comparison

This section first compares the final LSTM and LLM configurations on the dataset containing interpreted examinations, which was used as the classification dataset for Chapters 7 *LSTM Implementation* and 8 *LLM Implementation*. The models are then tested on the keyword-based version of this dataset, which was parse using the RAKE algorithm as described in Section 4.7 Information Extraction. We measure the model performance only with the general metrics, accuracy, precision, recall, F1 weighted and macro scores, with the F1 macro score being the primary deciding factor. Finally, we identify which dataset each algorithm performs best on, and these are examined in greater detail in Section 9.1.2 Independent Injury and Risk Assessments.

9.1.1 Dataset Performance

Here we elaborate on the previous hypothesis that input in the form of information structures instead of free text might benefit the Risk Predictor performance. Extracting keywords from the input reduces noise, e.g. non-essential and filler words or repeated phrases, and effectively captures the most vital information (Berry & Kogan, 2010). Using this algorithm, the text also resembles the original anamneses and examination (described in chapter 4 Study Design) in structure, because most were not fluent text, but rather key phrases separated with punctuation. However, in some cases keyword extraction possibly reduces the textual context and impairs the Risk Predictor's understanding.

Study Dataset

Table 9.1 shows that the LSTM model consistently outperforms BERT using the study dataset across all metrics, especially F1 macro score, which is our primary concern. The LSTM model's higher F1 macro score of 74% suggests that it balances precision and recall across all classes more accurately, making it the better choice for this particular task. These results confirm that LSTMs perform better on relatively small datasets and more specifically defined tasks. BERT possibly overfit and could have yielded better results with more data.

model	*accuracy*	$precision_w$	$recall_w$	$f1_w$	$f1_{mac}$
bert-final-interpreted	0.74	0.75	0.74	0.74	0.7
lstm-final-interpreted	0.78	0.78	0.78	0.78	0.74

Table 9.1: Performance comparison of LLM and LSTM Risk Predictors trained on the free-text study dataset

Keyword-Based Dataset

RAKE created the keyword-based dataset by subsetting the study dataset while trying to preserving the most vital information. According to Table 9.2, the LLM Risk Predictor slightly outperforms the LSTM model on the keyword-based dataset with an accuracy of 78% versus 77%, and F1 macro scores of 73% compared to 72%. BERT outperformed LSTM possibly because, even with the reduced information from keywords, BERT's architecture is better designed to capture and understand the remaining contextual relationships. Additionally, the keyword-based dataset likely removed some data noise, which could explain BERT performing better on this dataset compared to the broader study dataset.

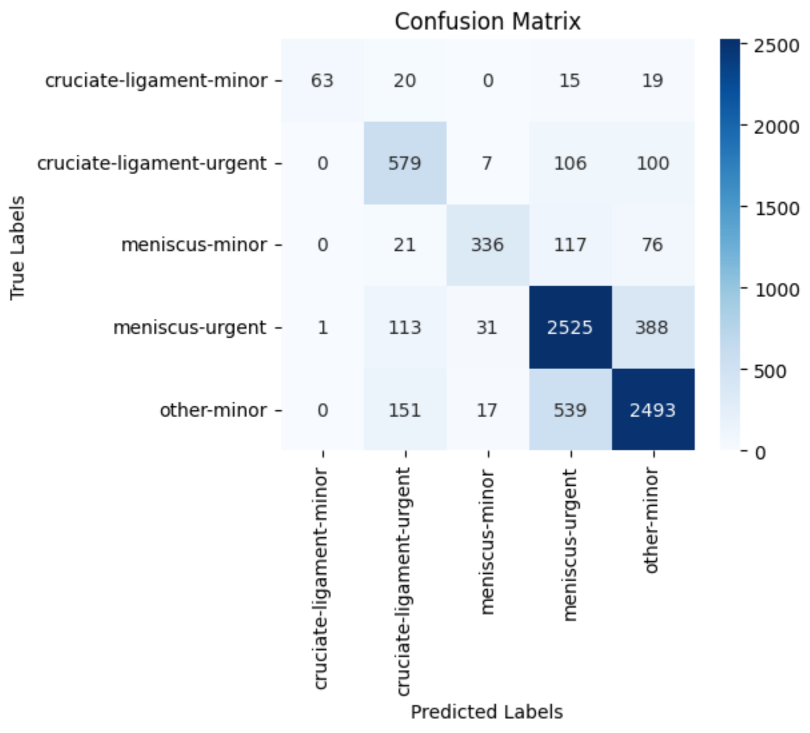
Evidently, the LSTM Risk Predictor using the study dataset outmatches all other algorithm-dataset pairs with accuracy, precision, recall and F1 weighted equal to 78% and F1 macro 74%. However, it was a closely contested evaluation, because BERT using the keyword-based data differs a mere 1% in F1 macro scores from the LSTM algorithm, with all other metrics remaining virtually identical. This 1% difference places BERT with the keyword-based dataset as the second-best model-dataset combination, just behind the LSTM on the study dataset.

model	*accuracy*	$precision_w$	$recall_w$	$f1_w$	$f1_{mac}$
bert-final-keywords	0.78	0.78	0.78	0.78	0.73
lstm-final-keywords	0.77	0.77	0.77	0.76	0.72

Table 9.2: Performance comparison of LLM and LSTM Risk Predictors trained on the keyword-based study dataset

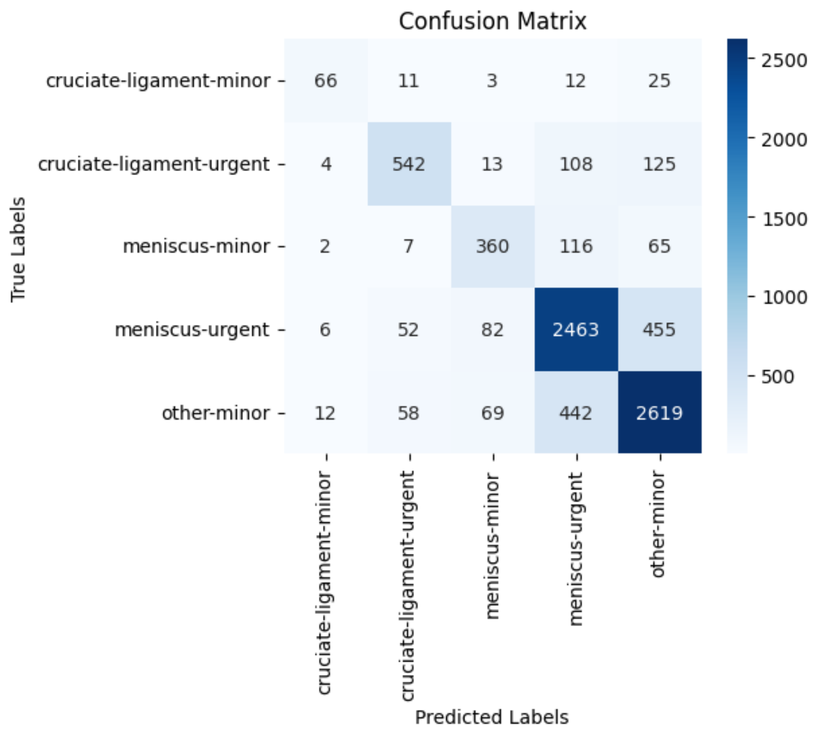
9.1.2 Independent Injury and Risk Assessments

The dataset labels consist of an injury type and risk level, for example *meniscus* (injury type) urgent (risk level). This label structure enables the Risk Predictor to predict both aspects simultaneously, summarizing patients' health state for the clinic staff and assisting in prioritizing the patients' injuries. However, this approach also prevents the model from demonstrating its ability to predict each aspect independently. Therefore, we evaluate the model's performance separately in predicting risk level, which is crucial for patient prioritization, and injury type, vital for further treatment. All computations are based on the confusion matrices of the LSTM Risk Predictor on the study dataset and the LLM (BERT) Risk Predictor on the keyword-based dataset in Figure 9.1 (a) and (b).



True \ Predicted	cruciate-ligament-minor	cruciate-ligament-urgent	meniscus-minor	meniscus-urgent	other-minor
cruciate-ligament-minor	63	20	0	15	19
cruciate-ligament-urgent	0	579	7	106	100
meniscus-minor	0	21	336	117	76
meniscus-urgent	1	113	31	2525	388
other-minor	0	151	17	539	2493

(a) LSTM



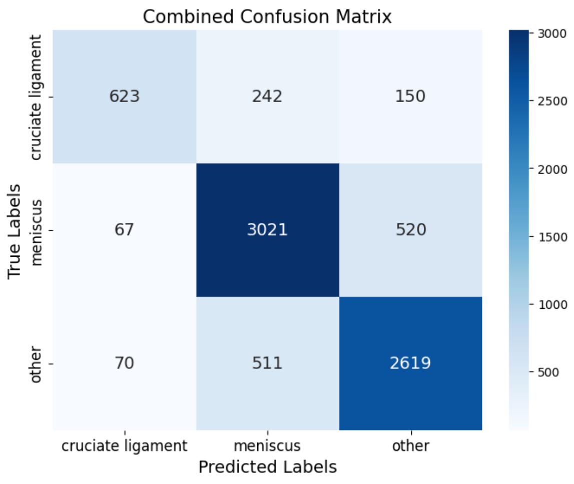
True \ Predicted	cruciate-ligament-minor	cruciate-ligament-urgent	meniscus-minor	meniscus-urgent	other-minor
cruciate-ligament-minor	66	11	3	12	25
cruciate-ligament-urgent	4	542	13	108	125
meniscus-minor	2	7	360	116	65
meniscus-urgent	6	52	82	2463	455
other-minor	12	58	69	442	2619

(b) LLM

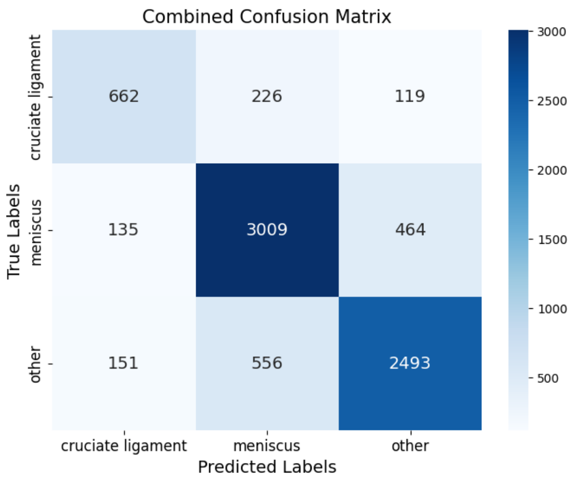
Figure 9.1: Comparison of classification results of the LSTM Risk Predictor, trained on the free-text study dataset, and LLM Risk Predictor, trained on keyword-based dataset

Injury type

The Risk Predictor algorithms categorize injuries into three main groups: meniscus, cruciate ligament, and other. To assess the performance of these algorithms on injury type alone, we recalculated the metrics accuracy, precision, recall, and both weighted and macro F1 scores using the confusion matrices of the LSTM and LLM Risk Predictors (shown in Figure 9.2) and aggregate them by summing the injury classes. For this evaluation, with the exception of the class *other minor*, which does not have an *urgent* counterpart, labels within the same injury category but with different risk levels were combined.



(a) LLM



(b) LSTM

Figure 9.2: Comparison of LSTM Risk Predictor (on free-text) and LLM Risk Predictor (on keywords) classification results for injury prediction

model	*accuracy*	$precision_w$	$recall_w$	$f1_w$	$f1_{mac}$
bert-keywords	0.8	0.8	0.8	0.8	0.78
lstm-keywords	0.79	0.79	0.79	0.79	0.76

Table 9.3: Performance comparison of LSTM Risk Predictor (on free-text) and LLM Risk Predictor (on keywords) for injury prediction

When the task does not include risk prediction, making it simpler, both BERT and the LSTM perform better (see Table 9.3). BERT benefits more from this simplification, achieving an F1 macro score of 78% compared to LSTM's 76%. Notably, BERT improved with 4% and the LSTM with 2%, compared to their results on the full classification. This performance boost highlights that both models struggle when required to predict both injury type and risk level, likely due to the complexity of the classification task nearly doubling.

Prioritization level

The risk predictor always assigns patients a label associated with a certain risk level (minor or urgent), based on this classification patients can be underprioritzed, correctly prioritized or overprioritized. Ensuring patients receive prompt care to prevent further complications is crucial, making correct prioritization the optimal outcome (Xie et al., 2021). However, if the model's prediction is not accurate, over-prioritization is still acceptable, whereas under-prioritization is considered suboptimal. Therefore, we introduce the metrics correct prioritization, underprioritization and overprioritization rate, and an overall priority score, which capture the patients' priority in a way that no other metric does.

Correct prioritization rate. *Correct prioritization rate*, also denoted as *CR* means the ratio of samples that the model assigned correctly with either minor, or urgent from all entries. It varies from 0 to 1, where 1 is the most optimal value. It differs from accuracy because the pair sample label *cruciate ligament urgent* and *meniscus urgent* is a real positive for this metric.

$$\text{Correct prioritization rate} = \frac{\text{Samples with correct risk}}{\text{All samples}}$$

Underprioritization rate. *Underprioritization rate*, shortened with *UR*, is the fraction of samples that the model assigned to a label with lower risk regardless of the injury type, for example a datapoint from the *cruciate ligament urgent* class received a label *other minor*. Its values range from 0 to 1, where 0 is the optimal value.

$$\text{Underprioritization rate} = \frac{\text{Samples, assigned a lower risk}}{\text{All samples}}$$

Overprioritization rate. Analogously to the underprioritization rate, the *overprioritization rate* (also marked as *OR*) represents the proportion of samples, which the model falsely classified as higher risk than they are, e.g. a sample from the *cruciate ligament minor* class receives a label *meniscus urgent*. Its values range from 0 to 1, where $1 - Correctprioritizationrate$ is the best value.

$$\text{Overprioritization rate} = \frac{\text{Samples, assigned a higher risk}}{\text{All samples}}$$

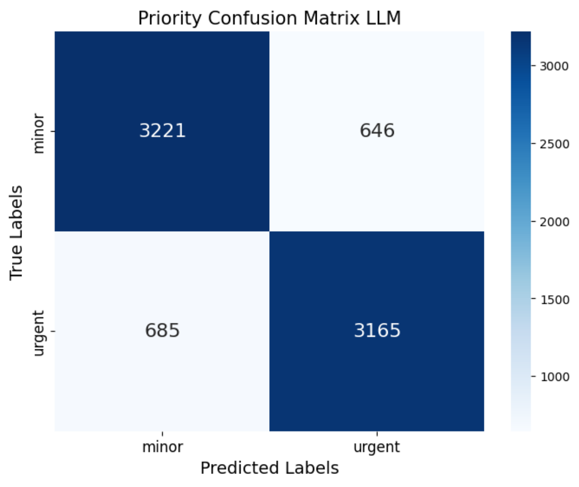
Priority score. The *priority score*, also called *Priority* weights the different prioritization rate types, described above, to show how effectively an algorithm prioritizes. It awards each correct prioritization with 1 point, overprioritization with 0.5 point and deducts 0.5 for underprioritization.

$$\text{Priority} = 1.CR + 0.5.OR - 0.5.UR$$

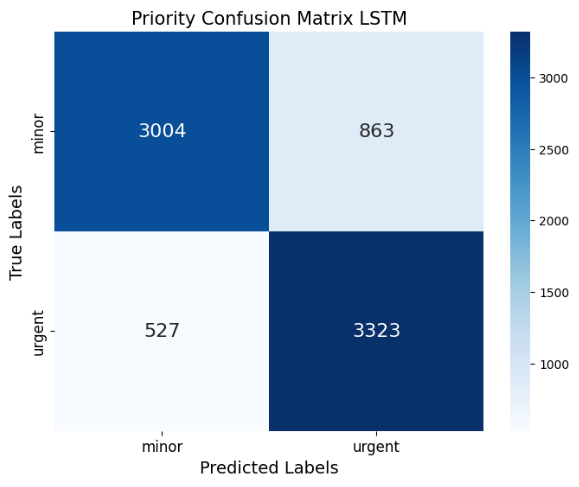
Where:

- CR - Correct prioritization rate
- OR - Overprioritization rate

- UR - Underprioritization rate



(a) LLM



(b) LSTM

Figure 9.3: Comparison of LSTM Risk Predictor (on free-text) and LLM Risk Predictor (on keywords) classification results for prioritization

model	*CR*	*OR*	*UR*	*Priority*
bert-priority	0.83	0.08	0.09	0.82
lstm-priority	0.82	0.11	0.07	0.84

Table 9.4: Performance comparison of LSTM Risk Predictor (on free-text) and LLM Risk Predictor (on keywords) classification results for prioritization

While BERT has a marginally better correct prioritization rate (CR) of 83% (as seen in Table 9.4), the LSTM Risk Predictor ultimately achieves a higher overall Priority score due to its lower underprioritization rate (UR) of 7%. We computed these values as described above with the help of the confusion matrices for both algorithms, which were aggregated by summing the samples by their risk level. Since underprioritization is particularly undesirable, LSTM's lower UR contributes significantly to its superior Priority score, making it the better model for this task in terms of balancing correct prioritization while minimizing the risks associated with underprioritization.

9.2 Threats to validity

This section evaluates factors that could potentially reduce the accuracy of the Risk Predictor in practical applications. First, we compare our Risk Predictor algorithms performance with and without the interpretation of the examinations to assess which vital information the interpretation process may be removing. Additionally, we compare the structure of the custom clinical dataset entries, collected via the iPad app, with the interpreted examination data points to discuss their similarities and differences. The Risk Predictor algorithms also perform a minimal test using the study dataset to practically highlight these potential threats to validity.

9.2.1 Interpretation validity

The uninterpreted data contains the unaltered anamnesis and examinations as made by the doctor, while the interpreted dataset has undergone filtering and rephrasing to remove any strong medical jargon or information that the patient has no way of knowing. As Chapter 4 Study Design explains, this pre-processing aims to simulate potential user queries, which also helps the model train more accurately. However, this pipeline naturally removes information that may help during the classification.

model	*accuracy*	$precision_w$	$recall_w$	$f1_w$	$f1_{mac}$
bert-final-original	0.79	0.79	0.79	0.79	0.74
lstm-final-original	0.8	0.8	0.8	0.8	0.75

Table 9.5: Performance comparison of LLM and LSTM Risk Predictors trained on the original data

Both the BERT and LSTM models perform better on the original, uninterpreted data across all metrics, including accuracy, precision, recall, F1 weighted, and F1 macro (as seen in Table 9.5 and Table 9.1). The performance decline is more pronounced in the LLM Risk Predictor, suggesting that the interpretation process may remove critical information that BERT relies on to make accurate predictions. LSTM, while also affected, shows a smaller decline, indicating it might be more robust to the interpretation process. Nonetheless, the decline is not overly significant and is expected, as the filtering of information is a necessary step, and the discarded information is likely not reflective of what would realistically be provided by a patient.

9.2.2 Questionnaire Dataset Evaluation

Our iPad app (detailed in Chapter 4 Study Design) collected 5 sets of patient responses to our patient questionnaire. We use these responses to linguistically compare them to the study dataset and evaluate how well the superior Risk Predictor, the optimized LSTM, performs on these entries, given that we have the confirmed patient diagnoses.

Linguistic similarities and differences

To establish how the two datasets align or differ, we examine their cosine similarity that accounts for word order and overlapping phrases. We also evaluate their readability, which indicates how complex a text is and how much context it provides. The readability is evaluated by the Flesch reading ease score, which puts into proportion the average sentence length in words and the average word length in syllables (Formula 9.1), resulting non-strictly in a number between 0 and 100 with 100 indicating great readability(Flesch, 1981).

$$206.835 - 1.015\left(\frac{\text{total words}}{\text{total sentences}}\right) - 84.6\left(\frac{\text{total syllables}}{\text{total words}}\right) \tag{9.1}$$

Figure 9.4 shows a comparison between two of the collected questionnaire and study dataset entries for the example patients with IDs 123 and 234. The average cosine similarity of the five entry pairs is 26.67%, which the figure exemplifies with the patient 123 entries having a score of 29% and for patient 234 - 4.74%. These low scores are most probably due to the study dataset texts being more fluent and coherent, with no duplicating information and no answers missing context (e.g. "Yes" in the questionnaire datapoint for patient 234) unlike their questionnaire counterparts. The lack of balance between details and context in the questionnaire answers potentially explains the study dataset entries having a mean readability score of 79.32% and the questionnaire data entries - 50.42%. These low similarity scores and the readability score discrepancies suggest that the final Risk Predictor is used to more contextual information and details, which lack in the questionnaire answers and thereby might experience difficulties correctly classifying the patient symptoms. Potential solutions to this problem could be to improve the questionnaire design to generate more fluent and non-repetitive answers, to use the Risk Predictors trained on the keyword-based datasets, whose fragmentation resembles the questionnaire answers or to retrain the Risk Predictors on interpreted data, which was reinterpreted to be less cohesive.

Test Performance

We tested how accurately the final Risk Predictor using LSTM classifies these five pairs of questionnaire and study dataset entries. As all of the patients had a *meniscus urgent* diagnosis, the results are possibly biased, however, the Risk Predictor has an accuracy of 100% on the study dataset and 60% on the questionnaire data. The entries that the Risk Predictor falsely assigned an *other minor* label are the example patients 123 and 234 in Figure 9.4. This misdiagnosis happened because patient 123 describes a Baker's cyst that could have been caused by arthritis, which has low risk in the study dataset (Leib et al., 2023), and patient 234's answer is very concise. The keyword-based LSTM Risk Predictor, however, diagnoses patient 234 also correctly.

Patient 123 (Cosine similarity: 29.0%)

Questionnaire dataset entry (Readability score: 32.77%)
63 Male Basically no acute event except now two weeks ago and last week an inflammation which was restored with cooling and ibuprofen otherwise two meniscusopses in 2005 and 2014 in 14 accompanied by a Baker's cyst which was not treated separately. The pain occurred again around 2016 Formation of a Baker's cyst and swelling of the Baker's cyst under severe working conditions and the joint was no longer as mobile The pain usually occurs in the buttock area as the Baker's cyst swells and the mobility of the knee is then no longer 100%. Sometimes there is also minimal pressure pain on the inside of the knee, which does not actually swell directly, but the VEKA cyst causes problems when walking if the Baker's cyst has enlarged, as already mentioned, the enlargement usually occurs when working a little harder or climbing a ladder or standing for a long time due to the strain. The complaints are then that the knee becomes stiff and I can no longer use baskets so well In the meantime, I not only have problems bending but also when fully stretching out due to the B cyst and the acute appearance problem As already mentioned before, two meniscus operations in 2005 and in 2014, the latter with Baker's cyst which was not surgically treated

Study dataset entry (Readability score: 73.88%)
63 years old 2005 and 2014 AC knee with IM partial resection in the left knee joint, symptom-free for 2 -3 years, increasing swelling of the back of the knee in the course of the operation, currently: max walking distance approx. 7 km, subsequently increasing. Baker's cyst Painkiller intake: occasionally. Ibuprofen Zn. Knee infection right knee joint. My left knee joint has no swelling, I can't hyperextend or fully extend it. Bending the knee is possible. My right knee joint also has no swelling, but there is pain near the inner side of the knee joint when I press it.

Patient 234 (Cosine similarity: 4.74%)

Questionnaire dataset entry (Readability score: 69.11%)
49 Male Got worse afterwards said position No Not putting cream on About three years ago Blocked once Yes No previous injuries

Study dataset entry (Readability score: 86.5%)
49 years old Z.n. AC knee right with Im TR 06/24. The patient copes well with the right knee postoperatively. Continued pain in the left knee joint with blocking sensation on flexion to the end, trauma denied. Complaints had been present for several years. So far, oral pain medication has been used for the right knee. No pinching sensations in the knee joint, pain particularly on the inside of the knee. I have no joint swelling and feel stable. There is pain on the inner side of the knee joint.

Figure 9.4: Comparison of questionnaire and study dataset entries for example patients with ID 123 and 234

10 Conclusion and Future Work

In this chapter, Section 10.1 summarizes our research and describes the AI algorithm configurations that we discovered to determine patient risk most accurately. Section 10.2 highlights general and specific to Long Short-Term Memory (LSTM) networks and Large Language Models (LLMs) aspects of our thesis that are worth investigating further.

10.1 Conclusion - AT

This thesis demonstrates the potential of Long Short-Term Memory (LSTM) networks and Large Language Models (LLMs) to prioritize high-risk patients with meniscus and cruciate ligament injuries. By interpreting medical data, such as physician anamneses and examinations, into descriptions of patient symptoms as if they were explained by the patients themselves, we created a study dataset, labelled with risk levels and injury localizations. Two Risk Predictors - one utilizing an LSTM algorithm and the other an LLM - trained on this dataset to be able to accurately classify injuries' type (meniscus, cruciate ligament, or other) and severity (urgent or minor). Both Risk Predictors were optimized through a series of training, parameter tuning, and structural enhancement techniques. We then evaluated their performance on the study dataset, both with and without parsing the data into keywords. The LSTM-based predictor achieved an accuracy of 78% and a macro F1 score of 74% on the study dataset, while the LLM-based predictor, using the keyword-parsed dataset, achieved an accuracy of 78% and a macro F1 score of 73%. Additionally, we propose a workflow for integrating the Risk Predictors into a clinical setting through an online patient portal. Although we acknowledge that the study dataset differs in language and structure from actual patient symptom descriptions, we believe that by carefully designing the web portal's questions and refining the training data, these differences can be minimized.

10.2 Future Work - JK, AT

In this section, we outline potential optimization possibilities for future investigation.

Enhanced data cleaning procedures could play a vital role in the system's future development, reducing noise and enabling the model to focus more effectively on relevant information. Additionally, the interpretation of anamneses, alongside examinations, should be considered as a priority, as these contain medical terminology not typically found in standard patient responses, which could enhance the model's performance with real patient data. Incorporating explainability features into the model is also crucial, as it would help build trust in the system and prove valuable in identifying and addressing potential biases.

Future improvements should focus on several key areas to enhance the overall system. It is important to add a privacy disclaimer to the web portal. Also developing a more user-friendly interface for the questionnaire portal is essential. A better-designed UI could encourage more patients to use the platform, thereby increasing user traffic and providing additional training data that could improve the accuracy of the risk prediction algorithm.

Another crucial area for enhancement is the integration of doctor feedback within the web portal. By allowing doctors to validate, confirm, or correct the predicted risks, the model could be retrained or continuously trained, further refining its accuracy over time. Expanding the model's prediction scope to cover a broader range of knee injuries or other medical conditions is another potential improvement. This expansion would require training on more diverse datasets to ensure the model can generalize effectively across different scenarios

For the LSTM model, several avenues for future work are suggested to enhance its performance. A deeper investigation into embeddings is necessary to find an optimal implementation, as this could significantly boost language understanding and overall model effectiveness. Despite the model's strong performance, there remains a noticeable divide between the test and training data, highlighting the need for further efforts to reduce overfitting. Regularization techniques should be explored and developed to address this issue.

Optimizing the learning rate is another critical step, with more tests needed to find the ideal rate and schedule for the model. This could help in mitigating overfitting and improving the model's performance. Additionally, experimenting with different optimizers beyond Adam—such as RMSprop, AdaGrad, or Nadam—might offer further improvements in model performance and convergence properties.

The LLM Risk Predictor showcases promising results in Chapter 9, although it remains inferior to the LSTM Risk Predictor in some evaluated categories. Exploring various optimization techniques further could directly mitigate these performance gaps and address the overfitting observed during training. However, a more compelling approach is to research the generation of detailed patient diagnoses using more advanced medical LLMs for the LLM Risk Predictor.

Generative LLMs like Llama have great potential to produce detailed patient diagnoses and determine their risk level. However, my free text generation baseline showed weaknesses as its results lacked uniformity and could not be properly validated. These problems could be addressed respectively using the prompt engineering technique few-shot prompting and utilizing professional clinical evaluation. Using few-shot prompts, similar to those in Figure 6.8, could guide the model to generate output in a desired format. This output formatting could help doctors navigate through the results better and potentially channel the LLM's attention on the correctness of the diagnosis.(Semnani et al., 2413) Additionally, working in an even closer collaboration with a group of clinicians, who could concretize the model's training data and give it feedback (similarly to Hirosawa et al., 2023), would ensure that the model produces medically correct data that adheres to the medical consensus with little patient harm.

Additionally to Med Llama 3 from Subsection 6.3.3, there are other even more advanced medical generative models available that are worth investigating. One such example is Med-PaLM, which leverages the PaLM 2 architecture (Chowdhery et al., 2022) and combination of prompt- and instruction-tuning. It is trained on general medical literature, clinical language, medical imaging, and genomic data, which provide the model with solid medical knowledge. It was also evaluated on how well it performed on a new composite metric MultiMedQA, which combines several datasets with multiple-choice medical exams and open-ended consumer questions. The results showed that the LLM's skills are comparable to those of a human clinician and ultimately made it the first AI system to pass a U.S. medical licensing exam. Moreover, Med-PaLM is suitable for medical NLP tasks other than question answering, such as classfication or medical assistance, with further training and fine-tuning (Singhal et al., 2023).However, as the model is developed by Google Research, developers can only access the model if they are affiliated with Google Cloud or belong to a research institution that was previously approved by Google. This limited access makes prevents general public access and slows the development process, but without any or minimal time contraints it is worth investigating.

Bibliography

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., et al. (2023). Gpt-4 technical report.

Ahmed, M., Seraj, R., & Islam, S. M. S. (2020). The k-means algorithm: A comprehensive survey and performance evaluation. *Electronics*, *9*(8), 1295.

Amari, S.-I. (1993). Backpropagation and stochastic gradient descent method.

Apicella, A., Donnarumma, F., Isgrò, F., & Prevete, R. (2021). A survey on modern trainable activation functions. *Neural Networks*, *138*, 14–32.

Arnold, C., Biedebach, L., Küpfer, A., & Neunhoeffer, M. (2024). The role of hyperparameters in machine learning models and how to tune them. *Political Science Research and Methods*.

Aßenmacher, M., Corvonato, A., & Heumann, C. (2021). Re-Evaluating GermEval17 Using German Pre-Trained Language Models.

Baldi, P., & Sadowski, P. J. (2013). Understanding Dropout. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, & K. Weinberger (Eds.), *Advances in Neural Information Processing Systems*. Curran Associates, Inc.

Berry, M. W., & Kogan, J. (Eds.). (2010). *Text Mining: Applications and Theory*. Wiley.

Blinov, P., Avetisian, M., Kokh, V., Umerenkov, D., & Tuzhilin, A. (2020). Predicting Clinical Diagnosis from Patients Electronic Health Records Using BERT-Based Neural Networks. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *12299 LNAI*, 111–121.

Boer, P.-T. D., Kroese, D. P., & Rubinstein, R. Y. (2005). A Tutorial on the Cross-Entropy Method. *Annals of Operations Research*, *134*, 19–67.

Bressem, K. K., Papaioannou, J.-M., Grundmann, P., & Borchert, F. (2023). MEDBERT.de: A Comprehensive German BERT Model for the Medical Domain.

Bruegge, B., & Dutoit, A. H. (2010). *Object Oriented Software Engineering: Using UML Patterns and Java*. Prentice Hall.

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: Synthetic Minority Over-sampling Technique.

Chomsky, N. (1957). *Syntactic Structures*.

Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., Schuh, P., et al. (2022). PaLM: Scaling Language Modeling with Pathways.

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

Church, K. W. (2017). Word2Vec. *Natural Language Engineering*, *23*(1), 155–162.

Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., Grave, E., Ott, M., Zettlemoyer, L., & Stoyanov, V. (2019). Unsupervised Cross-lingual Representation Learning at Scale.

Cooijmans, T., Ballas, N., Laurent, C., Gülçehre, Ç., & Courville, A. (2016). Recurrent batch normalization. *arXiv preprint arXiv:1603.09025*.

Cortes, C., Mohri, M., & Rostamizadeh, A. (2012). L2 regularization for learning kernels. *arXiv preprint arXiv:1205.2653*.

Devlin, J. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding.

Ding, B., Qian, H., & Zhou, J. (2018). Activation functions and their characteristics in deep neural networks. *2018 Chinese Control And Decision Conference (CCDC)*, 1836–1841.

Dobler, K., & de Melo, G. (2023). FOCUS: Effective Embedding Initialization for Monolingual Specialization of Multilingual Models.

Drummond, C., Holte, R. C., et al. (2003). C4. 5, class imbalance, and cost sensitivity: why under-sampling beats over-sampling. *Workshop on learning from imbalanced datasets II*, *11*(1–8).

Dumais, S. T. (2004). Latent semantic analysis. *Annual Review of Information Science and Technology*, *38*(1), 188–230.

Elkin, P. L., Liebow, M., Bauer, B. A., Chaliki, S., Wahner-Roedler, D., Bundrick, J., Lee, M., Brown, S. H., Froehling, D., Bailey, K., Famiglietti, K., Kim, R., Hoffer, E., Feldman, M., & Barnett, G. O. (2010). The Introduction of a Diagnostic Decision Support System (DXplain ™) into the workflow of a teaching hospital service can decrease the cost of service for diagnostically challenging Diagnostic Related Groups (DRG)s. *International Journal of Medical Informatics*, *79*(11), 772–777.

Esteva, A., Kuprel, B., Novoa, R. A., Ko, J., Swetter, S. M., Blau, H. M., & Thrun, S. (2017). Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, *542*, 115–118.

Fernández, S., Graves, A., & Schmidhuber, J. (2007). An Application of Recurrent Neural Networks to Discriminative Keyword Spotting. *4669*, 220–229.

Flesch, R. (1981). *How to Write Plain English*. Barnes & Noble.

Gers, F., & Schmidhuber, J. (2000). Recurrent nets that time and count. *Proceedings of the International Joint Conference on Neural Networks*, *3*, 189–194 vol.3.

Gers, F., Schmidhuber, J., & Cummins, F. (2000). Learning to Forget: Continual Prediction with LSTM. *Neural computation*, *12*, 2451–2471.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

Goodfellow, I. J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y. (2014). Generative Adversarial Networks.

Grandini, M., Bagli, E., & Visani, G. (2020). Metrics for Multi-Class Classification: an Overview.

Haixiang, G., Yijing, L., Shang, J., Mingyun, G., Yuanyue, H., & Bing, G. (2017). Learning from class-imbalanced data: Review of methods and applications. *Expert Systems with Applications*, *73*, 220–239.

Han, J., Kamber, M., & Pei, J. (2012). *Data Mining. Concepts and Techniques, 3rd Edition (The Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann.

Haykin, S. (1998). *Neural Networks: A Comprehensive Foundation* (2nd). Prentice Hall PTR.

Hirosawa, T., Kawamura, R., Harada, Y., Mizuta, K., Tokumasu, K., Kaji, Y., Suzuki, T., & Shimizu, T. (2023). ChatGPT-Generated Differential Diagnosis Lists for Complex Case–Derived Clinical Vignettes: Diagnostic Accuracy Evaluation. *JMIR Medical Informatics*, *11*.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*.

Huang, Z., Xu, W., & Yu, K. (2015). Bidirectional LSTM-CRF Models for Sequence Tagging.

Ioffe, S., & Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. In F. Bach & D. Blei (Eds.), *Proceedings of the 32nd International Conference on Machine Learning* (pp. 448–456). PMLR.

Jang, B., Kim, M., Harerimana, G., Kang, S. U., & Kim, J. W. (2020). Bi-LSTM model to increase accuracy in text classification: Combining word2vec CNN and attention mechanism. *Applied Sciences (Switzerland)*, *10*.

Johannessen, K. A., & Alexandersen, N. (2018). Improving accessibility for outpatients in specialist clinics: Reducing long waiting times and waiting lists with a simple analytic approach. *BMC Health Services Research*, *18*.

Jones, K. (1972). A Statistical Interpretation of Term Specificity in Retrieval. *Journal of Documentation*, *60*, 493–502.

Jordan, M., Blei, D. M., Ng, A. Y., & Edu, J. B. (2003). Latent Dirichlet Allocation Latent Dirichlet Allocation Michael I. Jordan. *Journal of Machine Learning Research*, *3*, 993–1022.

Kandel, E. R., Koester, J., Mack, S., & Siegelbaum, S. (2021). *Principles of neural science*. McGraw Hill.

Kaul, V., Enslin, S., & Gross, S. A. (2020). History of artificial intelligence in medicine.

Kudo, T., & Richardson, J. (2018). SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing.

Kulikowski, C. A., Kulikowski, S. M. W., & Weiss, C. A. (1982). Representation of Expert Knowledge for Consultation: The CASNET and EXPERT Projects.

Kuroiwa, T., Sarcon, A., Ibara, T., Yamada, E., Yamamoto, A., Tsukamoto, K., & Fujita, K. (2023). The Potential of ChatGPT as a Self-Diagnostic Tool in Common Orthopedic Diseases: Exploratory Study. *Journal of Medical Internet Research*, *25*.

Lample, G., & Conneau, A. (2019). Cross-lingual Language Model Pretraining.

Lee, J. Y., & Dernoncourt, F. (2016). Sequential Short-Text Classification with Recurrent and Convolutional Neural Networks.

Leib, A. D., Roshan, A., Foris, L. A., & Varacallo, M. (2023). Baker's cyst [PMID 28613525]. *StatPearls*.

Li, J., Sun, A., Han, J., & Li, C. (2020). A survey on deep learning for named entity recognition. *IEEE transactions on knowledge and data engineering*, *34*(1), 50–70.

Li, Z., & Arora, S. (2019). An exponential learning rate schedule for deep learning. *arXiv preprint arXiv:1910.07454*.

Lin, M., Chen, Q., & Yan, S. (2013). Network In Network.

Lipton, Z. C., Kale, D. C., Elkan, C., & Wetzel, R. (2015). Learning to Diagnose with LSTM Recurrent Neural Networks.

Liu, G., & Guo, J. (2019). Bidirectional LSTM with attention mechanism and convolutional layer for text classification. *Neurocomputing*, *337*, 325–338.

Liu, W., Wen, Y., Yu, Z., & Yang, M. (2016). Large-Margin Softmax Loss for Convolutional Neural Networks.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., & Stoyanov, V. (2019). RoBERTa: A Robustly Optimized BERT Pretraining Approach.

Lucas, P. J. F., & Gaag, L. C. V. D. (1991). Principles of Expert Systems.

M, H., & M.N, S. (2015). A Review on Evaluation Metrics for Data Classification Evaluations. *International Journal of Data Mining & Knowledge Management Process*, *5*, 01–11.

Manning, C. D., Raghavan, P., & Schütze, H. (2009). *An introduction to information retrieval*. Cambridge University Press.

McCarthy, J., Minsky, M. L., Rochester, N., & Shannon, C. E. (2006). A Proposal for the Dartmouth Summer Research Project on Artificial Intelligence.

Mihalcea, R., & Tarau, P. (2004). Textrank: Bringing order into text, 404–411.

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient Estimation of Word Representations in Vector Space.

Mohammed, R., Rawashdeh, J., & Abdullah, M. (2020a). Machine Learning with Oversampling and Undersampling Techniques: Overview Study and Experimental Results. *2020 11th International Conference on Information and Communication Systems (ICICS)*, 243–248.

Mohammed, R., Rawashdeh, J., & Abdullah, M. (2020b). Machine Learning with Oversampling and Undersampling Techniques: Overview Study and Experimental Results. *2020 11th International Conference on Information and Communication Systems, ICICS 2020*, 243–248.

Neumann, M., King, D., Beltagy, I., & Ammar, W. (2019). ScispaCy: Fast and Robust Models for Biomedical Natural Language Processing. *Proceedings of the 18th BioNLP Workshop and Shared Task*.

Ni, L., Lu, C., Liu, N., & Liu, J. (2017). Mandy: Towards a smart primary care chatbot application (J. Chen, T. Theeramunkong, T. Supnithi, & X. Tang, Eds.), 38–52.

Nielsen, J. (1989). Usability Engineering at a Discount.

Niu, Z., Zhong, G., & Yu, H. (2021). A review on the attention mechanism of deep learning. *Neurocomputing*, *452*, 48–62.

O'Beirne, J., O'Dwyer, T., O'Rourke, J., & Quinlan, W. (1984). The diagnosis of knee injuries in casualty–a prospective study. *Injury-international Journal of The Care of The Injured*, *15*(4), 232–235.

Openai, A. R., Openai, K. N., Openai, T. S., & Openai, I. S. (2018). Improving Language Understanding by Generative Pre-Training.

O'shea, K., & Nash, R. (2015). An introduction to convolutional neural networks. *arXiv preprint arXiv:1511.08458*.

Pascanu, R., Mikolov, T., & Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In S. Dasgupta & D. McAllester (Eds.). PMLR.

Peeperkorn, M., Kouwenhoven, T., Brown, D., & Jordanous, A. (2024). Is Temperature the Creativity Parameter of Large Language Models?

Popel, M., & Bojar, O. (2018). Training Tips for the Transformer Model.

Prechelt, L. (2000). Early Stopping - But When?

Pritchard, J. K., Stephens, M., & Donnelly, P. (2000). Inference of Population Structure Using Multilocus Genotype Data. *Genetics*, *155*(2), 945–959.

Quinn, J., McEachen, J., Fullan, M., Gardner, M., & Drummy, M. (2020). *Dive into deep learning: Tools for engagement*. Corwin, a SAGE Company.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019a). Language Models are Unsupervised Multitask Learners.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019b). Language Models are Unsupervised Multitask Learners.

Ruder, S. (2016). An overview of gradient descent optimization algorithms.

Rumelhart, D. E., Hinton, G. E., & Williams, R. J. (1988). Learning internal representations by error propagation.

Sæther, S. M. M., Sæther, S. M. M., Heggestad, T., Heimdal, J.-H., Heimdal, J.-H., & Myrtveit, M. (2019). Long Waiting Times for Elective Hospital Care - Breaking the Vicious Circle by Abandoning Prioritisation. *International journal of health policy and management*, *9*(3), 96–107.

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2019). DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter.

Schmidt, M., Fung, G., & Rosales, R. (2009). Optimization methods for l1-regularization. *University of British Columbia, Technical Report TR-2009-19*.

Schuster, M., & Nakajima, K. Japanese and Korean Voice Search. In: IEEE, 2012, 5460. ISBN: 9781467300469.

Semnani, S. J., Yao, V. Z., Zhang, H. C., & Lam, M. S. (2413). WikiChat: Stopping the Hallucination of Large Language Model Chatbots by Few-Shot Grounding on Wikipedia.

Sennrich, R., Haddow, B., & Birch, A. (2015). Neural Machine Translation of Rare Words with Subword Units.

Sharma, S., Sharma, S., & Athaiya, A. (2017). Activation functions in neural networks. *Towards Data Sci*, *6*(12), 310–316.

Shimodaira, H. (2000). Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, *90*, 227–244.

Shortliffe, E. (1977). Mycin: A Knowledge-Based Computer Program Applied to Infectious Diseases*. *Proceedings / the ... Annual Symposium on Computer Application [sic] in Medical Care. Symposium on Computer Applications in Medical Care*.

Singhal, K., Azizi, S., Tu, T., Mahdavi, S. S., Wei, J., Chung, H. W., Scales, N., Tanwani, A., Cole-Lewis, H., Pfohl, S., Payne, P., Seneviratne, M., Gamble, P., Kelly, C., Babiker, A., Schärli, N., Chowdhery, A., Mansfield, P., Demner-Fushman, D., . . . Natarajan, V. (2023). Large language models encode clinical knowledge. *Nature*, *620*, 172–180.

Smith, L. N. (2017). Cyclical learning rates for training neural networks. *2017 IEEE winter conference on applications of computer vision (WACV)*, 464–472.

Strauss, E. J., Day, M. S., Ryan, M., & Jazrawi, L. (2016). Evaluation, treatment, and outcomes of meniscal root tears: a critical analysis review. *JBJS reviews*, *4*(8), e4.

Su, J., Lu, Y., Pan, S., Murtadha, A., Wen, B., & Liu, Y. (2021). RoFormer: Enhanced Transformer with Rotary Position Embedding.

Sun, C., Qiu, X., Xu, Y., & Huang, X. (2019). How to Fine-Tune BERT for Text Classification?

Suzuki, K. (2011). *Artificial neural networks: methodological advances and biomedical applications*. BoD–Books on Demand.

Team, L., & Meta, A. @. (2024). The Llama 3 Herd of Models.

Thorndike, E. L. (1931). *Human learning*. The Century Co.

Tinabo, R., Mtenzi, F., & O'Shea, B. (2009). Anonymisation vs. Pseudonymisation: Which one is most useful for both privacy protection and usefulness of e-healthcare data, 1–6.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., & Lample, G. (2023). LLaMA: Open and Efficient Foundation Language Models.

Vasiliev, Y. (2020). *Natural language processing with Python and spaCy: A practical introduction*. No Starch Press.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention Is All You Need.

Vujović, Ž. (2021). Classification Model Evaluation Metrics. *International Journal of Advanced Computer Science and Applications*, *12*, 599–606.

Waibel, A., Hinton, G., Shikano, K., & Lang, K. J. (1989). Phoneme Recognition Using Time-Delay Neural Networks.

Wang, T., Roberts, A., Hesslow, D., Le Scao, T., Chung, H. W., Beltagy, I., Launay, J., & Raffel, C. (2022). What language model architecture and pretraining objective works best for zero-shot generalization? *PMLR*.

Wang, Y., Huang, M., Zhu, X., & Zhao, L. (2016). Attention-based LSTM for aspect-level sentiment classification.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, *35*, 24824–24837.

Werbos, P. J. (1990). Backpropagation through time: what it does and how to do it.

Wu, J., Chen, X.-Y., Zhang, H., Xiong, L.-D., Lei, H., & Deng, S.-H. (2019). Hyperparameter Optimization for Machine Learning Models Based on Bayesian Optimizationb. *Journal of Electronic Science and Technology*, *17*(1), 26–40.

Wythoff, B. J. (1993). Backpropagation neural networks A tutorial.

Xie, F., Zhou, J., Lee, J. W., Tan, M., Li, S., Rajnthern, L. S., Chee, M. L., Chakraborty, B., Wong, A.-K. I., Dagan, A., et al. (2021). Benchmarking emergency department triage prediction models with machine learning and large public electronic health records.

Xu, Z., Jain, S., & Kankanhalli, M. (2024). Hallucination is Inevitable: An Innate Limitation of Large Language Models.

Yu, Y., Si, X., Hu, C., & Zhang, J. (2019). A review of recurrent neural networks: Lstm cells and network architectures.

Zhong, T., Zhao, J., Guo, X., Su, Q., & Fox, G. (2023). RINAS: Training with Dataset Shuffling Can Be General and Fast.

Zhou, C., Sun, C., Liu, Z., & Lau, F. C. M. (2015). A C-LSTM Neural Network for Text Classification.

Zou, H., & Hastie, T. (2005). Regularization and variable selection via the elastic net.