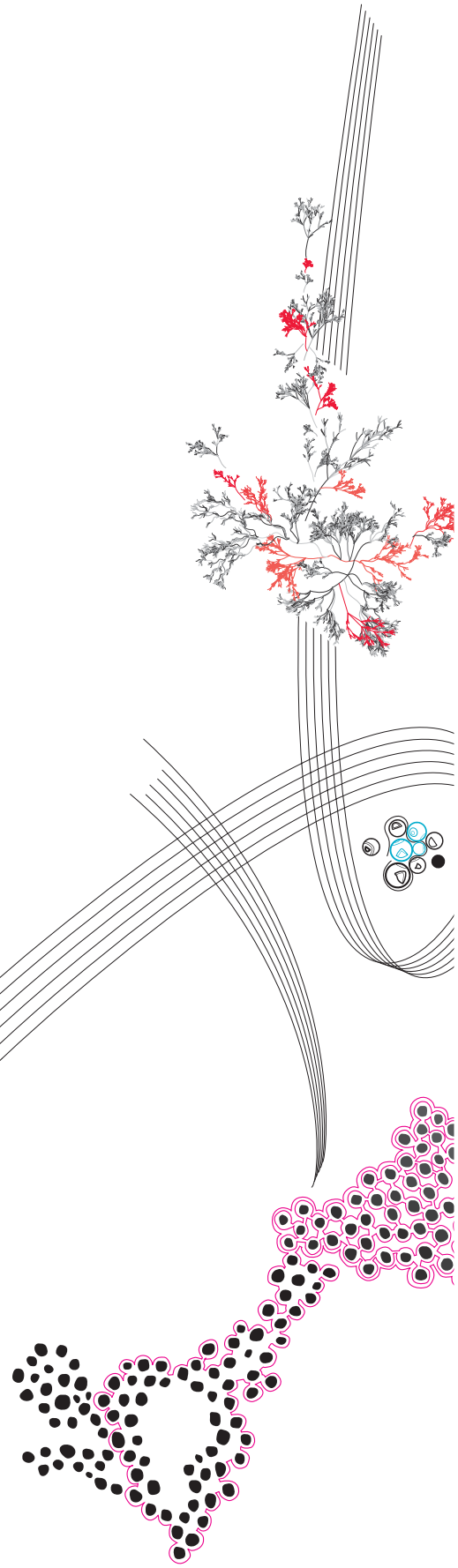


MSc Computer Science
Final Project

Generating natural language triage advice at Dutch out-of-hours primary care centers

Gijs Koehorst

Supervisors:
dr. Shenghui Wang (University of Twente)
dr. Faiza A. Bukhsh (University of Twente)
ing. Rob de Negro (Topicus)
dr. Marleen M. Nieboer (Topicus)

December, 2023

Department of Computer Science
Faculty of Electrical Engineering,
Mathematics and Computer Science,
University of Twente

UNIVERSITY OF TWENTE.

Acknowledgements

First of all, I would like to thank Topicus for providing me with the opportunity to conduct this research. In particular, special thanks to Rob de Negro and Marleen Nieboer for their guidance. The freedom they gave me, along with their interest in my findings, kept me motivated throughout the process.

Furthermore, I would like to thank Shenghui Wang for the discussions during our bi-weekly meetings. Her critical questions pushed me forward. I would also like to express my gratitude to Faiza Bukhsh for her interest in my work.

Also, a big thank you to everyone from the "huisartsenpost" of Arnhem who, in one way or another, contributed to this research. Without your data and participation in the user study, this research would not have been possible.

Lastly, I would like to thank my family for always supporting me in my studies, regardless of the path I chose.

Abstract

When a patient calls an out-of-hours primary care center, a triagist goes through different steps to assess the urgency of the situation. These steps are entered into a software system. This software is provided by Topicus under the name VIPLive Spoed EPD, for a large part of the Dutch out-of-hours primary care centers. Triagists are faced with high workloads at these centers. As such, there is value in reducing their workload. In many cases, triagists provide an advice which is recorded as natural language. In this study, we aimed to generate this advice based on historical data. The generated advice could be presented as a suggestion, which could reduce the typing workload of triagists.

Given their great success in various natural language generation (NLG) tasks, this study aimed to apply pre-trained language models to tackle this problem. Different models were finetuned and their performance was compared using ROUGE scores.

A healthcare-specific Med2Med and domain-generic Rob2Rob model were proposed. Rob2Rob was found to outperform Med2Med, showing that healthcare-specific is not necessarily beneficial in this situation.

Our data consisted of natural language and categorical features. We proposed an All-Text approach that simply concatenates all features as plain text using a separator token. We found that including the categorical features in this approach, improved the generated advice. Therefore, we also proposed a Fuse-Early approach that first processes the categorical features separately before fusing them with natural language features. This approach was found to hurt the performance, showing that the proposed All-Text was superior in this situation.

To provide more input context, different strategies for concatenating the features in the All-Text approach were proposed. The best strategy was found to be introducing a new unique token for each feature into the tokenizer of the Rob2Rob model.

To assess the usefulness of the proposed Rob2Rob model in practice, a user study was conducted which involved 4 triagists and 3 general practitioners. This study showed that the Rob2Rob is capable of generating appropriate advice in the majority of cases. However, domain experts expressed their doubts about the usefulness of the model outputs in practice.

The same user study also aimed to assess the usefulness of an explainability method that was proposed, which is based on integrated gradients. We found that the majority of domain experts did not find this method useful as it was found to be distracting and occasionally marked irrelevant words.

Given the small sample size of the user study, it could be valuable to conduct a pilot study where the model is actually integrated into the VIPLive Spoed EPD application, to really assess the usefulness of the Rob2Rob model in practice.

Keywords: triage advice, language models, NLG, feature fusion, integrated gradients

Contents

Chapter 1

Introduction

The Nederlandse Triage Standard (NTS) [1], which translates to Dutch Triage Standard, is a standard for triage in the chain of acute care. Triage refers to the dynamic process of determining urgency and deciding on subsequent actions. The standard is intended for emergency department nurses, triage nurses at out-of-hours primary care centers (OOH-PCCs), and emergency medical dispatchers at ambulance control centers. The goal of the NTS is to increase the consistency and efficiency of triage in acute care, so that patients receive appropriate care and treatment as quickly as possible from the right healthcare professional. The patient and their care needs are the central focus.

When a patient calls an out-of-hours primary care center, a triagist goes through different steps, outlined by the NTS, to assess the urgency. These steps are entered into a software system. This software is provided by Topicus under the name VIPLive Spoed EPD, for a large part of the Dutch OOHPCCs.

1.1 Motivation

A triage is supposed to be quick and due to an ageing population and a general shortage of medical staff, there is a high workload in Dutch healthcare. This is also the case for OOHPCCs as reported in Dutch news [56, 21]. Therefore, there is value in reducing reporting workload of triagists. This reduction can be achieved by minimizing the amount of typing required by a triagist. This may be realised using Natural Language Generation (NLG) techniques that are able to generate text that otherwise would have to be typed. By leveraging historical data, NLG can be used to provide educated suggestions that support triagists.

NLG is one of the two sub-fields of Natural Language Processing (NLP). Previous work has shown the application of NLP techniques in the healthcare domain. Most research focused on classification tasks using natural language input, which is a form of Natural Language Understanding (NLU). These classification tasks include the prediction of the medical specialties at hospital admission [3, 32], early detection of dementia [6], predicting in-hospital cardiac arrest or intensive care unit admission [10], classification of Multiple Sclerosis severity [13], early prediction of lung cancer [17], predicting emergency department disposition [50, 65] and extracting the diagnostic goal of an MRI scan of Multiple Sclerosis patients [44]. On the other hand, NLG has been applied in healthcare to reduce the overall workload in various areas. For example, hospital documents [19, 30] have been summarized to allow healthcare professionals to access information in a shorter amount of time. Question and answering systems have been developed [25, 28, 43, 53] to support/replace healthcare professionals in less critical situations. NLG techniques have also been applied in an

attempt to directly reduce the reporting workload of healthcare professionals. For example, medical imaging reports have been generated [29]. Others [20] developed an auto-completion system that determines when a clinician intends to input a clinical concept and provides suggestions at those specific points. Previous work [37] also generated electronic health records consisting of natural language, where the input was a combination of text and tabular data. However, this previous work on generating health records did not leverage pre-trained language models (PLMs), which are fundamental to state-of-the-art (SOTA) approaches in various NLG tasks, such as machine translation [42] and text summarization [64]. These PLMs are pre-trained on a very large corpus in an unsupervised way by for example masking tokens and training the model to retrieve those tokens. This way a PLM has knowledge of the language it was pre-trained on. Subsequently, they can be finetuned in a supervised way on a specific task, while benefiting from the already present knowledge. Therefore, it is interesting to explore whether these PLMs are useful for NLG in a triage setting.

In the Spoed EPD application of Topicus, the triage starts with free-form textual input where the problems of the patient are recorded. This is followed by multiple categorical inputs, where selections are made out of a predefined set of options. Finally, depending on the urgency and next step, the triagist potentially writes free-form textual advice. This advice is given based on previous steps. Therefore, it is interesting to explore whether PLMs can be used to generate this text advice.

However, there are potential challenges that come with this. First of all, PLMs for NLG expect both the input and output to be text and are therefore not traditionally designed to handle categorical data, such as the symptoms that the triagist selects out of a predefined set of options. Nevertheless, there could potentially be valuable information in this categorical data that the model can use when generating an output. Secondly, triage data may contain domain specific language and a distinct writing style. PLMs are usually trained on publicly available online text, like Wikipedia. However, they might not extensively incorporate language specific to particular domains, such as healthcare. To tackle this problem, several PLMs have been trained on healthcare-specific language [34, 26, 35, 39], including one in Dutch [55]. In some previous work these domain-specific models outperformed domain-generic models on certain tasks [34, 26], while in other work they underperformed [44]. Thirdly, evaluating the generated text is not straightforward, since it may not be definitively right or wrong, and there is no benchmark for comparison. Even if the generated text appears syntactically similar to the text that was originally written by a healthcare professional, there could still be errors present. Finally, PLMs are black box models, which makes it difficult to understand why a certain output is given. This could make it difficult for healthcare professionals to trust the output of the model. To mitigate this problem, explainability approaches have been proposed. These approaches are used to make the decision-making processes of a model more understandable to humans.

1.2 Research questions

The main goal of this thesis is to explore whether PLMs can be leveraged to generate free-form textual advice of triagist that would otherwise have to be typed during triages at Dutch out-of-hours primary care centers. Therefore, also given the motivation and challenges outlined in section 1.1, we formulate the following main research question and corresponding sub-questions:

> ***RQ**: How can PLMs be applied to automatically generate advice of a triagist during a triage of an out-of-hours primary care center?*

- ***SQ1**: What is the performance of a domain-specific PLM against a domain-generic PLM?*
  This question aims to determine whether healthcare-specific pre-training is beneficial over domain-generic pre-training when using PLMs to generate text in a triage setting.

- ***SQ2**: How can natural language and categorical features be combined to train a PLM that generates triage advice?*
  Categorical features are not traditionally used as input for language models. This question aims to determine how we can fuse these features with the natural language features and to see if they have added value in terms of model performance.

- ***SQ3**: What strategies with regard to the setup of the PLM can be used to improve its performance to generate advice?*
  A language model can have different setups during training and inference. Therefore, different setups may lead to differences in the model's performance to generate advice.

- ***SQ4**: How can we evaluate the generated advice?*
  This question aims to determine how we can evaluate whether the outputs of the trained language model are valid and useful in practice.

- ***SQ5**: How can we add explainabilty to the generated advice?*
  This question aims to determine how we can explain the generated advice of the model to the end-user.

1.3 Structure of the Thesis

The remaining of this thesis is structured in the following way. Chapter 2 contains background information on (pre-trained) language models. Chapter 3 discusses related work on the use of PLMs in the medical domain, the use of multi-modal input data, decoding strategies, evaluation methods and model explainability. Chapter 4 introduces the dataset and the cleaning steps that were performed. Chapter 5 presents the proposed methodologies such as the models that were used. Chapter 6 reports on all the experiments that were carried out to answer our research questions. Chapter 7 presents the user study that was conducted to evaluate our model. Chapter 8 concludes this thesis and discusses limitations and future work.

Chapter 2

Background

This chapter provides the background knowledge for this thesis. Section 2.1 introduces the concept of a conditional language model, which is used in this work to generate triage advice. Section 2.2 introduces the encoder-decoder architecture and gives an overview of specific encoder-decoders such as RNNs and transformers, which form the basis of the PLMs used in this research. Finally, section 2.3 gives a brief introduction to PLMs, which are leveraged in this work.

2.1 Conditional language model

A conditional language model assigns probabilities to a sequence of words $W = (w_1, w_2, ..., w_n)$ given some context C, which for example consists of the input text. So, it models the probability distribution $P(W|C)$. If we have a model with some parameters θ, then the goal is to find:

$$\arg\max_{W} P_\theta(W|C) \tag{2.1}$$

Using the chain rule we can decompose this to:

$$P_\theta(w_1, w_2, ..., w_n|C) = \prod_{t=1}^{n} P_\theta(w_t|C, w_1, w_2, ..., w_{t-1}) \tag{2.2}$$

Where $w_1, w_2, ..., w_{t-1}$ represent the previously generated words. This defines a generative model that can generate words in an autoregressive manner. So, at each step t a conditional language model produces a probability distribution p over all possible next words. This probability is penalized for being different from the true distribution $\hat{p}$, which is simply a one-hot vector representing the actual next token. Cross-entropy can be used to measure the difference between these two probability distributions:

$$L(\hat{p}, p) = -\hat{p}log(p) \tag{2.3}$$

Since $\hat{p}$, is a one-hot vector we have:

$$L(\hat{p}, p) = -log(p_{w_t})) = -log(p(w_t|w_1, w_2, ..., w_{t-1}, C)) \tag{2.4}$$

Therefore, we can train a conditional language model by minimizing the per word negative log-likelihood:

$$L(\theta) = -\frac{1}{T}\sum_{t=1}^{T} logP_\theta(w_t|w_1, w_2, ..., w_{t-1}, C) \tag{2.5}$$

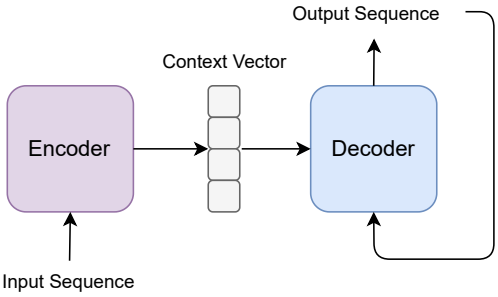


FIGURE 2.1: High level encoder-decoder architecture

Note that the loss is computed based on the actual previous tokens. This is called teacher forcing [59], where the model is forced to base its predictions on correct previous words. This should lead to more stable training and faster convergence.

2.2 Sequence-to-Sequence models for NLG

Traditional Deep Neural Networks (DNNs) require that the inputs and targets are encoded by vectors of fixed dimensions. Therefore, they are not suitable for tasks where the inputs and targets are sequences of variable length, which is the case in sequence-to-sequence tasks. So instead, sequence-to-sequence models are used. Sequence-to-sequence models often come in the form of an encoder-decoder architecture. As the name implies, this architecture consists of an encoder and a decoder. The encoder is responsible for processing the input sequence and transforming it into a fixed-length representation, which is commonly referred to as the context vector. This vector represents the contextual information of the entire input sequence. The decoder in turn uses the context vector to produce an output sequence in an autoregressive manner. It also takes into account the tokens that it has already generated. A high-level overview of the encoder-decoder architecture is given in figure 2.1. In the following subsections we discuss different implementations of the encoder-decoder architecture.

2.2.1 Vanilla Recurrent Neural Networks

At first, these architectures consisted of two simple Recurrent Neural Networks (RNNs) [11], one for the encoder and one for the decoder as shown in figure 2.2. Each element of the input sequence is fed to the encoder one by one. At each step a hidden state h_t is computed based on the previous hidden state h_{t-1} and the current input element x_t:

$$h_t = g_1(W_{hh}h_{t-1} + W_{hx}x_t + b_h) \tag{2.6}$$

Where b_h is the bias for the hidden state and W_{hh} and W_{hx} are the weights for the previous hidden state and the current input respectively. g_1 is an activation function, like for example $tanh$. When the encoder has processed the final element of the sequence, we end up with a final hidden encoder state. This encoder state functions as a context vector representing the input sequence and is used as the initial hidden state of the decoder. The hidden state of the decoder h_t is calculated based on the previous hidden state h_{t-1} and previous output y_{t-1}:

$$h_t = g_2(W_{hh}h_{t-1} + W_{hy_{t-1}}y_{t-1} + b_h) \tag{2.7}$$

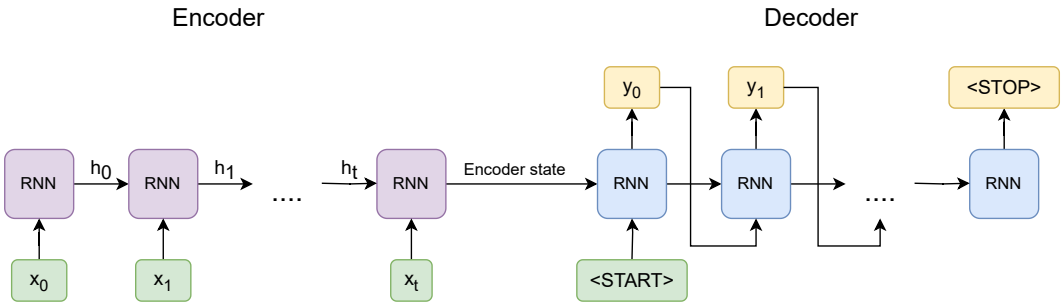


FIGURE 2.2: Encoder-decoder architecture with vanilla RNNs

Where b_h is the bias for the hidden state and W_{hh} and $W_{hy_{t-1}}$ are the weights for the previous decoder hidden state and the previous decoder output respectively. g_2 is again an activation function. The decoder computes an output based on the new hidden state h_t:

$$y_t = g_3(W_{hy}h_t + b_y) \tag{2.8}$$

Where b_y is the bias for the output and W_{hy} are the weights for the current decoder hidden state and the output. g_3 is again an activation function. This process continues until the decoder produces a stop token for y_t, which marks the end of the output sequence. Due to vanishing/exploding gradients [7], and the fact that all memory is being compressed into a single context vector of fixed size, RNNs struggle with longer sequences.

2.2.2 Long Short-Term Memory

To mitigate the short term memory problems of vanilla RNNs, Long Short-Term Memory (LSTM) cells [23] were introduced into the recurrent architecture [52], which should lead to less forgetting of initial elements. Each RNN cell of the unfolded encoder-decoder architecture of figure 2.2 is replaced by an LSTM cell, which is depicted in figure 2.3. First, the current input x_t and previous hidden state h_{t-1} go through a forget gate which determines which fraction of the long term memory is retained in the cell state c:

$$f_t = \sigma(W_{hf}h_{t-1} + W_{xf}x_t + b_f) \tag{2.9}$$

Then, the current input x_t and previous hidden h_{t-1} state go through the input gate. Here the candidate long term memory q_t is computed:

$$q_t = tanh(W_{hq}h_{t-1} + W_{xq}x_t + b_q) \tag{2.10}$$

A sigmoid function of this input gate determines what fraction of c_t is actually remembered long term:

$$u_t = \sigma(W_{hu}h_{t-1} + W_{xu}x_t + b_u) \tag{2.11}$$

So, the new cell state c_t is computed based on the output of the forget and input gate:

$$c_t = c_{t-1}f_t + u_t c_t \tag{2.12}$$

Finally, the output gate updates the short-term memory. The new cell state c_t is passed through a $tanh$, to determine the potential new short-term memory, and the current input

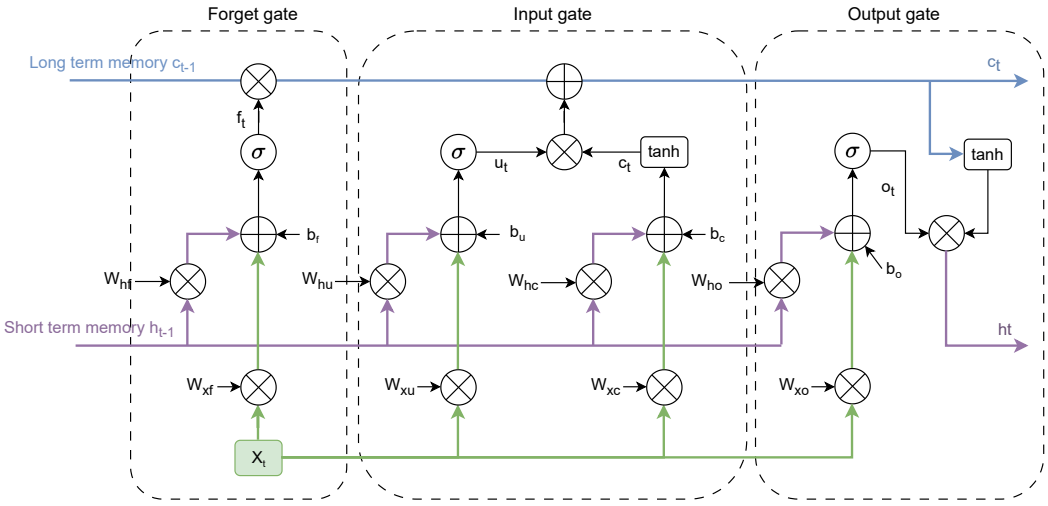


FIGURE 2.3: LSTM architecture

x_t and previous hidden h_{t-1} again go through a sigmoid, to determine how much of that potential short-term memory is passed on as the new short-term memory h_t:

$$h_t = \sigma(W_{ho}h_{t-1} + W_{xo}x_t + b_o)tanh(c_t) \tag{2.13}$$

As can be seen in figure 2.3 and equation 2.12, the cell state c itself does not go through any weight operations, therefore it does not suffer from vanishing/exploding gradients. Thus, compared to vanilla RNNs, it deals better with longer sequences. However, the short term memory h is still exposed to vanishing/exploding gradients.

Another potential problem in the described encoder-decoder architecture, is that the encoder bases the hidden states only on previous context. It can be beneficial for the encoder to also look to future context. This can be achieved with bidirectional RNNs [46], where the input is passed once from left to right and once from right to left. The same concept applies to bidirectional LSTMs, since they are still RNNs. In sequence-to-sequence models only the encoder can be bidirectional, because the decoder autoregressively generates a sequence and therefore cannot look into the future.

2.2.3 Attention mechanism

There is still a bottleneck problem, where all memory has to be compressed into one context vector of fixed length. Therefore, also LSTMs still struggle with long sequences. An attention mechanism [4] can be introduced to mitigate this problem. This mechanism has different forms, but the main idea is that the context vector (encoder output) is given access to all hidden states instead of only the final hidden state. At each decoder step a context vector is calculated by taking a weighted sum over all encoder hidden states:

$$c_i = \sum_{j=1}^{x} a_{ij}h_j \tag{2.14}$$

Where, c_i is the context vector at decoder step i, h_j is j-th encoder hidden state, x is the number of encoder hidden states and a_{ij} is the attention weight for encoder hidden state j at decoder step i. The attention weight determines how much attention the decoder should pay at step i to encoder hidden state j. These weights can for example be learned using

a fully connected linear layer, which takes in the previous decoder output $i-1$ and all encoder hidden states $[h_1..h_x]$ and outputs an attention score for each encoder hidden state h_j. Using a Softmax, these attention scores are converted to attention weights a_{ij} that are used to calculate the context vector in equation 2.14. The model can learn the weights of the introduced fully connected layer. Therefore, it can learn to which encoder hidden states it should attend, given the encoder hidden states and the previous decoder output.

2.2.4 Transformers

In 2017, Vaswani et al. [54] introduced the transformer architecture, which uses an attention mechanism called self-attention. They showed that the self-attention mechanism in itself is powerful enough, and that recurrence is not necessary when there is enough data. This means that the tokens of the input sequence do not depend on each other anymore, which allows for parallel computation, leading to decreased training time.

The architecture of a transformer is depicted in figure 2.4. The inputs are word embeddings where words are represented by vectors. Positional encoding is added to these inputs. This is required, since there is no recurrence in this architecture, so there would be no information about (relative) token position, without positional encoding. The word embeddings are fed through a Multi-Head self attention block. Self attention allows each word in the input to attend to all other words in the input. This way the model can learn to associate certain words with other words. In contrast to LSTMs, transformers do not struggle with really long sequences, because the self-attention mechanism has a potentially infinite reference window, given infinite computing resources. The self attention mechanism is depicted in figure 2.5. Word embeddings of the input sequence are fed through 3 different fully connected linear layers resulting in Query Q, Key K and Value V vectors. The dot product between Q and V results in a attention score matrix, where each combination of words has a score which represents how much those words should attend to each other. The scores are scaled down by dividing by $\sqrt{d_k}$, where d_k is the dimension of Q and V. This should lead to more stable gradients [54]. The Softmax of these scores is computed resulting in attention weights. Finally, the dot-product between V and the attention weights is taken.

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.15}$$

If the attention weight of a word is higher, this word is considered more important. To make this MultiHead attention (as shown in figure 2.4), Q, K and V are split into N vectors, which leads to N Heads. Each split goes through the self attention process shown in figure 2.5 individually. The intuition behind having multiple heads is that each head can learn something different by attending to information of different sub-spaces. The individual outputs of the heads are concatenated before going through a fully-connected linear layer. The result is processed through a feed forward fully-connected network, which marks the end of the Encoder part of the architecture.

The decoder part of the architecture receives the encoder output as well as previous decoder outputs. The previous outputs are positionally encoded in the same way as the inputs of the encoder. They then go through Masked Multi-Head Attention. This mechanism is the same as shown in figure 2.5 apart from one difference. After scaling, the attention scores are masked, so that the decoder cannot condition on future words. This is necessary because the decoder is autoregressive and generates an output sequence token by token. The output of this Masked Multi-Head Attention block functions as V for the other Multi-Head Attention block in the decoder. The output of the encoder functions as K and

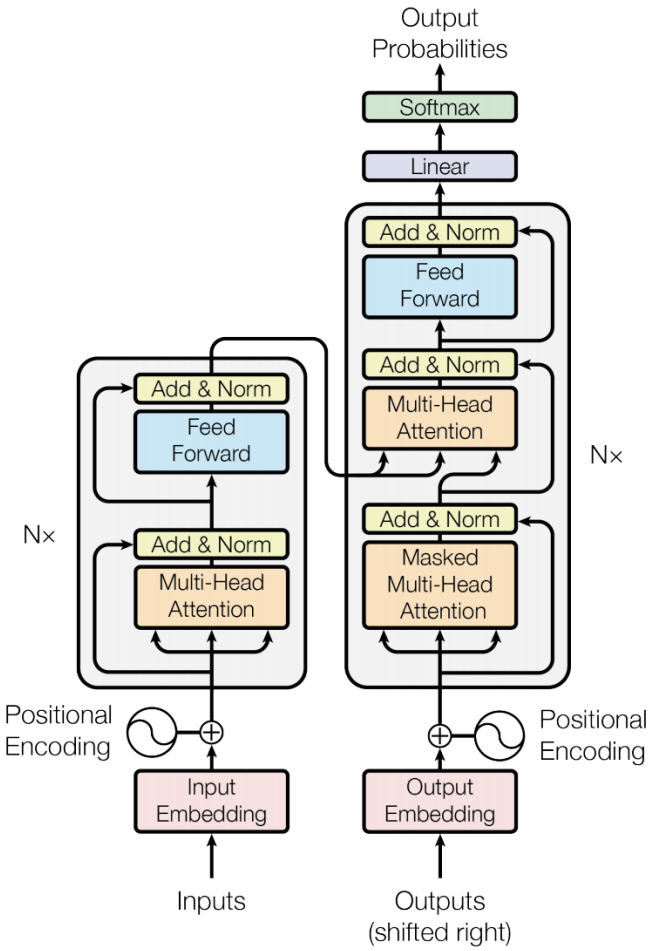


FIGURE 2.4: Transformer architecture from [54]

Q. This enables the decoder to decide, which encoder input should be attended to the most. From there, this Multi-Head Attention block works the same as the one in the encoder. Also here, the output is fed through a fully connected feed forward network. The output of this network goes through a final linear layer and Softmax, resulting in probability scores for each word in the vocabulary. The decoder outputs the word with the highest probability. This process is repeated until the decoder outputs a stop token.

Each block in the encoder and decoder also has a residual connection. The residual connection skips a Multi-Head Attention block or Feed Forward network and is then simply added to the output of that block. This way it produces a signal which is not affected by the vanishing gradient problem, which mitigates this problem to some extent.

2.3 Pre-trained language models

Deep neural networks, that are often used in NLP, are known for their large number of parameters. Consequently, the availability of labeled data is often a limiting factor in the performance of such models. To mitigate this problem, PLMs were introduced. These models are pre-trained on large corpora in an unsupervised manner. They are, for instance, trained to fill missing words in sentences. These PLMs have achieved great results in NLP

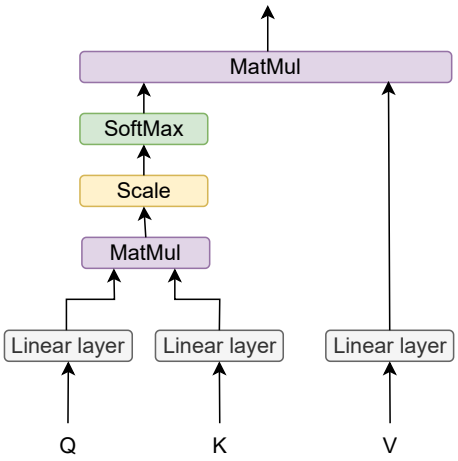


FIGURE 2.5: Self-attention edited from [54]

when they are finetuned on a downstream task in a supervised way. Therefore, this has lead to a paradigm shift from supervised learning to unsupervised pre-training followed by finetuning on a downstream task [57].

In theory any language model can be pre-trained. However, most PLMs use transformer architectures, since these allow for parallelization. This significantly speeds up training, which enables pre-training on huge corpora. In general, we can distinguish three different transformer architectures that form the basis for PLMs:

- *Encoder-only* architectures consist of a bi-directional encoder that is trained by recovering randomly masked tokens in the input sequence. They are traditionally used for Natural Language Understanding (NLU) tasks.
- *Encoder-Decoder* architectures consist of a bi-directional encoder and an autoregressive unidirectional decoder. They can be pre-trained by randomly masking tokens at the source, and retrieving those tokens at the target. They are traditionally used for a combination of NLU and NLG, in particular sequence-to-sequence tasks.
- *Decoder-only* architectures consist of a uni-directional decoder and can be pre-trained by autoregressively predicting tokens. They are traditionally used for NLG tasks.

Chapter 3

Related Work

This chapter introduces related work. First, an overview is given of the use PLMs in the medical domain. Then related work that use multi-modal data as input is introduced. Furthermore, it presents different decoding strategies that are used when generating text with PLMs. Also, different evaluation metrics used in NLG are presented. Finally, related work on attribution methods is introduced.

3.1 PLMs in medical domain

In the medical domain, BERT [16] has so far been the most used PLM. It is for example used to create embeddings of clinical text, which are then further processed through a neural network to predict whether a patient will develop dementia [6]. Others fine-tuned a BERT model to predict, based on unstructured text, to which medical specialty a patient should be referred [32]. D'costa et al. [13] used BERT in combination with a CNN and classification layer to predict the multiple sclerosis severity of patients.

However, PLMs have not been leveraged for NLG in the medical domain. Previous work [43] has used encoder-decoder architectures with BiLSTMs to create mental healthcare chatbots. They compared greedy and beam search and found that a beam search decoder produced significantly more empathetic responses with higher precision in terms of BLEU-score. Liu [37] used the vanilla transformer to generate electronic health records. This transformer was not pre-trained, but its components do form the basis for PLMs. The multi-modal input of the model consisted of structured and unstructured text such as demographic data, medication information, lab test information and past medical notes. They used an All-Text approach, where the different inputs were concatenated into a single string with special delimiter tokens. The goal of the model was to predict the sequence of a clinical note, based on some note-context that consists of past data extracted from an Electronic Health Record, a note type and a hint of 10 tokens of the note that is currently being written. They showed that commonly-used templates in the dataset were learned and reproduced by the transformers. They evaluated the models using perplexity per token, accuracy of next token and ROUGE-1 and ROUGE-2. However, no evaluation by domain experts was done.

Domain-specific models have shown benefits over domain-generic models, for example in text-mining tasks [34] and predicting 30-day readmission using discharge summaries [26]. Currently, the only Dutch, healthcare-specific PLM is MedRoBERTa.nl [55], which was pre-trained on nearly ten million hospital notes.

Elfrink et al. [17] adopted two RoBERTa [38] based architectures to predict lung cancer using Dutch medical notes: MedRoBERTa.nl and the domain-generic PLM RobBERT [15].

The performance of these models was comparable for this classification task.

Rietberg et al. [44] used the domain-specific, Dutch, model MedRoBERTa.nl and the domain-generic, Dutch, models BERTje [14] and RobBERT [15] to classify unstructured reports on their diagnostic goal. They showed that BERTje outperformed the other models, which shows that a domain specific PLM does not have to be superior for certain tasks.

Dutch healthcare-specific PLMs have, to our knowledge, not been used to generate text. MedRoBERTa.nl is encoder-only and is therefore not traditionally used for NLG tasks. However, Rothe et al. [45] showed that these models can successfully perform various NLG tasks when they are used as both the encoder and decoder in an encoder-decoder architecture. This requires two adjustments to the model when it is used in the decoder: masking the self-attention mechanism to hide the right context and adding an encoder-decoder attention mechanism.

3.2 Multi-modal input

Electronic health records usually consist of both structured (tabular) and unstructured data (unstructured text). Therefore, this data is multi-modal. PLMs traditionally expect the input to be a sequence. However, there are ways to incorporate tabular data into the input. For example, T5 [42], a PLM trained to perform a variety tasks, took a naive approach to incorporate structured data by simply casting the values to a text format, such that all tasks were converted to text-to-text.

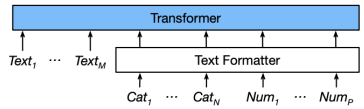
Various fusion strategies exist to incorporate categorical data, but in general we can distinguish All-Text, Fuse-Early and Fuse-Late [47], which are depicted in figure 3.1. The All-Text approach simply converts the numeric and categorical variables to strings, before concatenating them to the unstructured text, resulting in a string that contains all modalities. In Fuse-Early an embedding is learned for each categorical and numerical field. These embeddings are then jointly fed through a transformer. This strategy to embed categorical variables is for example used in the TabTransformer architecture [27], which is designed to model tabular data using contextual embeddings. The Fuse-Late approach first processes each modality separately through networks and only aggregates them near the end using pooling methods.

In the experiments of Shi et al. [47], the Fuse-Late approach gave the best scores on average, but the differences were minor. We also have to note that all of their tasks were classification or regression. The Fuse-Late approach does not make sense for NLG tasks, because the strategy fuses near the output layer. In a NLG task this output layer has to be the decoder, which means it would simply be a Fuse-Early strategy as depicted in figure 3.1.

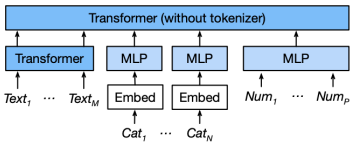
To our knowledge, apart from the previously discussed work of Liu [37], there is no previous work that fuses text and categorical data for a NLG task in the medical domain. Furthermore, apart from the discussed work by [47], different fusion strategies are rarely compared. Therefore, next, we will also discuss previous work in the medical domain that uses multi-modal data, but not necessarily for NLG.

Zhang et al. [65] used a "reason for visit" text field to improve models that predict whether someone is admitted into the hospital. They used bag-of-words in combination with principal component analysis (PCA) to get a representation of the free text field. These representations were combined with structured data about a patient to train logistic regression and neural network models. Including the free text representations resulted in an improvement in model performance.

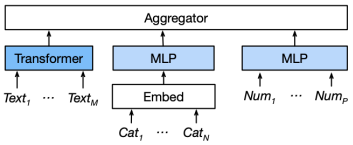
Arnaud et al. [3] used triage notes to make an early prediction on the medical specialities



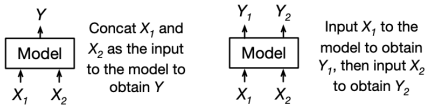
(a) *All-Text*. Convert numeric and categorical values into additional text tokens.



(b) *Fuse-Early*. Transformer operates on learned embeddings for each feature.



(c) *Fuse-Late*. Separate branches encode each modality, and aggregated via concatenation.



(d) Notation used in these figures.

FIGURE 3.1: Fusion strategies for unstructured text and tabular data from [47]

at hospital admission. They used a Fuse-Late approach, where a MLP and a CNN were trained in parallel on structured data and unstructured triage notes respectively. The resulting outputs were concatenated and fed through another MLP. The inclusion of unstructured triage notes boosted the performance.

Chen et al. [10] combined numerical representations of unstructured clinical narratives and structured clinical and demographic data, before feeding it through various machine learning models to predict possible critical outcomes. Including the clinical narratives boosted performance. The model outperformed emergency physicians in the prediction task.

Jing et al. [29] used a CNN in combination with co-attention and a hierarchical LSTM to automatically generate medical imaging reports.

All of this previous work that uses multi-modal data in the medical domain [3, 10, 29], do not leverage PLMs and are therefore potentially missing out of performance, given that they are SOTA in many NLP/NLG tasks.

3.3 Decoding strategies

Text generated by conditional language models can be far from perfect. The quality of the generated text is partly determined by the decoding strategy. The simplest strategy is Greedy search, which simply outputs the token with the highest probability at each time step. Therefore, posterior tokens with high probabilities may be hidden. So, it may not always lead to an optimal sequence.

Beam search aims to mitigate this problem by keeping track of the n most likely beams and finally choosing the beam with the highest overall probability. However, decoding strategies that attempt to find high likelihood sequences have been found to generate undesired text that is incoherent and gets into repetitive loops [62, 24].

Other decoding strategies were proposed to generate higher quality text. Top-K sampling [18] involves the filtering of the K most probable candidate words for the next sequence element, followed by a redistribution of the probability mass solely among these K candidate, which should lead to more human-sounding text. However, because K is fixed, problems can occur when some words are sampled from a sharp distribution, whereas others from a flat distribution.

Therefore, Nucleus Sampling [24] was proposed. It selects the minimum set of words whose cumulative probability surpasses a given probability threshold p, rather than confining

the selection to the K most probable words. The probability mass is then redistributed among this word set. This allows for dynamic variation in the size of the word set, which is determined by the distribution of probabilities for the next word.

Welleck et al. [58] argue that repetitive word sequences are caused by the model training approach. Their findings indicate that, based on human evaluations, beam search can generate more fluent text than Nucleus Sampling, given the modification of the model's training objective.

3.4 Evaluation methods for text generation

Various standard metrics, such as accuracy and F1, are used to evaluate the performance of machine learning models. These metrics are straightforward in for example classification problems, because the number of possible outputs is fixed. So, the prediction is either correct or incorrect. This is not the case for models that generate text.

Therefore, other metrics, such as ROUGE [36] and BLEU [40] and METEOR [5] were introduced. These measures give a score based on the number of matching n-grams between the generated text and the label. Therefore, these scores say something about the syntactical similarity between the prediction and the label. However, a limitation of these metrics is that even if a generated piece of advice is syntactically similar to a reference piece of advice, a subtle difference could mean the generated advice is incorrect.

More recently, PLMs like BERTScore [63] and BARTScore [61] have been adopted to score the predictions based on semantic similarity. By including semantics, these scores should be more similar to human judgements compared to syntactic scores. However, a subtle difference in the generated advice could still make it incorrect. Additionally, calculating these scores is resource intensive due to the use of PLMs.

3.5 Model explainability

Explaining why language models give certain outputs is difficult given the size and complexity of the architectures. Nevertheless, explainability is important, especially for adoption of such a model in practice. Calculating input attributions is a commonly used method to try to explain model outputs. Different feature attributions methods have been proposed. For example, the product of gradients and feature values [49] can be used as a measure of feature importance for a certain output. Other methods, such as DeepLIFT [48] and Layer-wise relevance propagation (LRP) [8] use back-propagation based approaches, where the prediction is back-propagated through the network towards the individual features. However, Sundararajan et al. [51] argue that these gradient and propagation-based approaches violate two fundamental axioms for attribution methods. The first axiom is Sensitivity, which states that if an input and baseline differ in one feature but have a different output, then the feature with different values should receive a non-zero attribution. The second axiom is Implementation Invariance, which states that if two networks with different implementations yield identical outputs for all inputs, then the input attributions should also be identical. To satisfy these axioms they proposed an attribution method called Integrated Gradients [51]. This method defines a baseline input x', for example a zero vector, from which a straight-line path is interpolated to the actual input x. This path is divided into several steps and the gradients are computed for each step. Integrated gradients are derived by accumulating these gradients. So, integrated gradients can be described as the integral of gradients along a direct line from the baseline x' to the input x. For a network F, this can

be approximated for feature i of input x and baseline x' as follows:

$$IG_i(F, x, x') = (x_i - x'_i) \sum_{j=0}^{m} \frac{\partial F(x' + \frac{j}{m}(x - x'))}{\partial x_i} \frac{1}{m} \tag{3.1}$$

where m is the number of steps along the straight-line path from x' to x. Attribution methods cannot be directly applied to language generation, because there is not a single output. However, it can be used to calculate the attributions of each input and previously generated output token for every output token.

3.6 Research Directions

In the medical domain, PLMs have predominantly been employed for NLU tasks. Domain-specific PLMs have also been developed and utilized, but their performance varies compared to domain-generic PLMs. Also, there is a scarcity of research on the use of PLMs for NLG tasks specifically in this domain. Moreover, the exploration of utilizing multi-modal data as input for PLMs in this area is also lacking. Additionally, there is limited research on the various fusion strategies for integrating such multi-modal data. Furthermore, there are various evaluation metrics for NLG, but they have their limitations in assessing the appropriateness of generated advice. Also, decoding strategies can impact the performance of language models. Finally, explainabality methods have been proposed to explain model outputs. Therefore, this leaves the following research directions when generating the advice of a triagist:

1. Exploring domain-specific versus domain-generic PLMs in a triage setting.
2. Exploring the usefulness of including categorical data as input to PLMs in a triage setting.
3. Exploring different fusion strategies when including categorical data as input to PLMs in a triage setting.
4. Exploring how we can evaluate the usefullness of our fintuned PLM.
5. Exploring explainability methods to make the decision-making processes of the finetuned PLMs more understandable.

Chapter 4

The Data

In this chapter, we will cover the available dataset. First, we describe the process that produced the data. Then we will describe the dataset and its fields, followed by an analysis of the data. Additionally, we will outline the cleaning steps that were performed. Finally, we will discuss the data that will be used as input for our models.

4.1 Process that produces the data

When a patients calls an out-of-hours primary care center, a triagist assesses the urgency based on the following steps:

1. ABCD-check: is there a risk to life? ABCD stands for airway, breathing, circulation and disabilities
2. Subjective DA: What is the problem of the patient? In the application of Topicus this is represented by the Subjectief Doktersassistent (SDA) field. This field is filled by the triagist and contains the subjective problems of the patient. Although this is a unstructured text field, there is some structure in it consisting of 5 points:
    - Klacht/beloop: what problems is the person, who calls/visits, experiencing?
    - Hulpvraag: what is the care question of the person calling/visiting?
    - Voorgeschedenis: history
    - Medicatie: medication
    - Algemeen: general points
3. The triagist selects the entry symptoms from a pre-defined list of options.
4. Based on these symptoms the triagist is presented with a set of questions. For each question the triagist can select an answer from a pre-defined set.
5. Determine urgency: based on triage criteria and triage type (physical or telephone triage). The triage nurse can adjust the urgency based on context, warning signs, and risk groups. The possible urgency levels are:
    - $U0$: loss of ABCD - resuscitation
    - $U1$: unstable ABCD - immediate danger to life (immediate)
    - $U2$: threat to ABCD or organ damage (as soon as possible)
    - $U3$: realistic chance of harm/human reasons (within a few hours)

- $U4$: negligible chance of harm (within a day)
- $U5$: no chance of harm (next working day)

6. The triagist determines the next step out of a set of options:
   - AMBU: ambulance
   - SEH: Spoedeisende hulp (emergency center)
   - VLK behandelaar
   - HA: Huisarts (visit General Practitioner)
   - Politie: Police
   - FastTrack
   - Advies: Advice
7. If the Urgency level is U4 or U5, the triagist gives advice to the patient. This can either be advice selected from a pre-defined list or tailored advice as unstructured text.

After every triage that is carried out, a general practitioner who is present at the call center goes through the following steps:

1. **Subjective GP**: the patient indicates what he/she is suffering from. This step is usually copied by the GP from the SDA field of the triagist.
2. **Objective**: objective observations in the case the next step is HA (visit the GP).
3. **Evaluation**: making an evaluation based on the Subjective and Objective step.
4. **ICPC**: International Classification of Primary Care, which is a standard for encoding and classifying complaints, symptoms, and conditions in primary care.
5. **Plan**: a plan is made for the patient. If the next step of the triage was an advice, the GP usually copies the advice from the triagist.

4.2 Dataset

The data comes from a single Dutch out-of-hours primary care center. It includes information of triages that took place between 2015 and 2023. The data consists of 901902 rows. Each row represents a conducted triage. The fields in table 4.1 represent the data that is produced by triagists during triages. `sda_regel` contains different sub-fields inside the unstructured text. Therefore, we extracted these sub-fields(`klacht_beloop`, `hulpvraag`, `voorgeschiedenis`, `medicatie`, `algemeen`) into separate fields using regular expressions. Their description is given in table 4.3. The fields in table 4.2 represent what is entered by a GP after a call.

4.3 Data analysis

Given that many fields are optional, the data has many null values. Figure 4.1 shows the fraction of null values for each column. We see from this figure that `aanvullendadvies`, which is the unstructured text advice of triagist that we want to predict, is missing in many

TABLE 4.1: Fields entered by triagist during call. Whether a field is optional is in this case based on the NTS [1].

Name	Description	Data type	Optional	Notes
`begintijd`	Start date and time of the triage	DateTime	No	
`eindtijd`	End date and time of the triage	DateTime	No	
`sda_regel`	unstructured text entered by the triagist describing the symptoms, questions of the patient and patient history	String	No	Some pre-filled structure present
`ingangsklachten`	Symptoms of the patient	Enum	No	
`vragen`	Questions to be asked based on `ingangsklachten`	Enum	No	Possible values determined by `ingangsklachten`
`antwoorden`	Answers to the questions of `vragen`	Enum	No	Possible values determined by `vragen`
`nts_urgentie`	Triagist can deviate from proposed urgency	Enum	No	Possible values based on `ingangsklachten` and `antwoorden`
`nts_urgentie_gekozen`	Urgency level chosen by the triagist	Enum	No	Is the same as `nts_urgentie` if the triagist does not deviate
`nts_vervolgstap`	Next step for the patient proposed by the system based on the urgency	Enum	No	System automatically selects it based on the `urgency`
`nts_vervolgstap_gekozen`	Next step for the patient chosen by the triagist	Enum	No	Is the same as `vervolgstap` if the triagist does not deviate
`gekozen_advies`	Advice chosen by triagist	Enum	Yes	
`aanvullend_advies`	Advice given by triagist in unstructured text form	String	Yes	

TABLE 4.2: Fields filled by GP after a call

Name	Description	Data type	Optional	Notes
`sha_regel`	unstructured text entered by GP describing the symptoms, questions of the patient and patient history	String	No	Possibility to copy `sda_regel` from triagist
`o_regel`	Objective observations by GP	String	Yes	Only applicable for physical triage
`e_regel`	Evaluation by GP	String	No	
`p_regel`	Treatment plan for the patient	String	No	Advice by triagist considered

TABLE 4.3: Fields extracted from `sda_regel`

Name	Description	Data type	Optional
`kacht_beloop`	Symptoms and course of those symptoms as described by the patient	String	No
`hulpvraag`	Help request of the patient	String	Yes
`voorgeschiedenis`	Medical history of the patient	String	Yes
`medicatie`	Medication the patient is taking	String	Yes
`drugs_alcohol`	Whether the patient took any drugs or alcohol	String	Yes

cases. This is because this advice is usually only written when `nts_vervolgstap`, which is the next step, is "Advies" which is an advice from the triagist.

In figure 4.2 we see that "Advies" is chosen less than "HA". This explains the lack of filled advice fields, since "HA" means the patient should visit the center, which means an advice by the triagist is not necessary. Figure 4.3 confirms this, since U3 usually leads to "HA", while U5 leads to an advice by the triagist.

The following points are the result of analysing the unstructured text fields:

- The `aanvullendvies` sometimes starts with an abbrevation like "iom reg arts", which means the advice follows from a discussion the triagist had with the GP that is present in the call center.
- Depending on the date of the triage, COVID can have an impact on the contents of the text field, since they for example checked whether a person had COVID symptoms

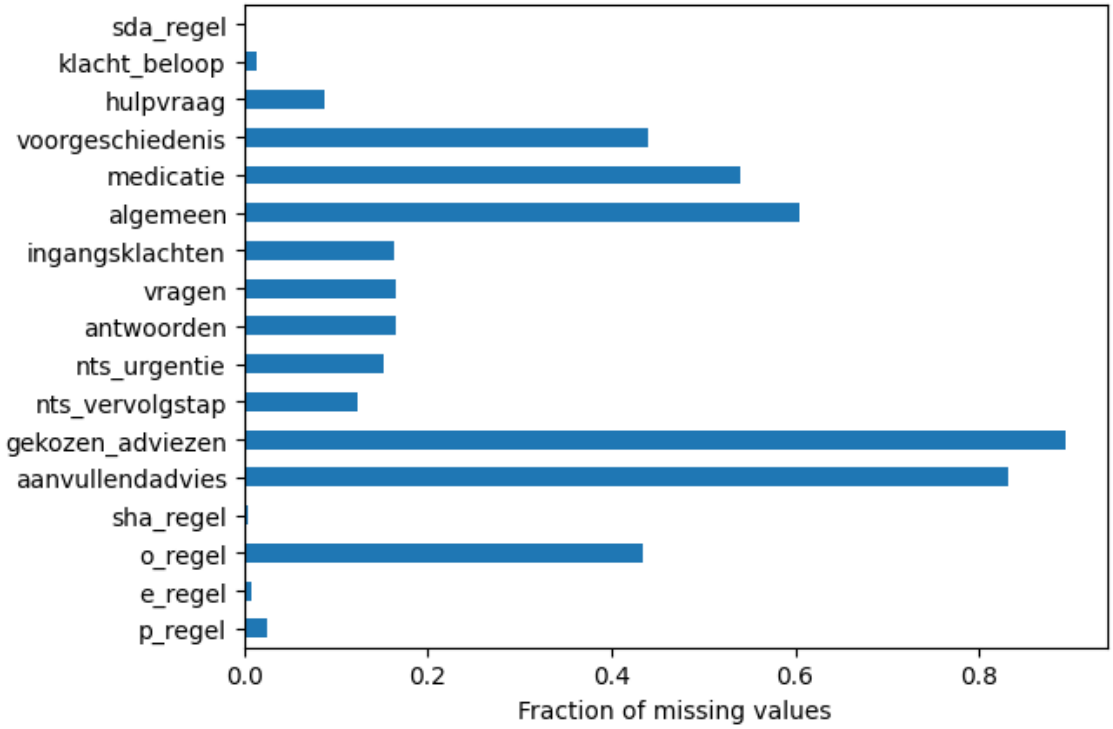


FIGURE 4.1: The fraction of rows that has no value for each column of the data

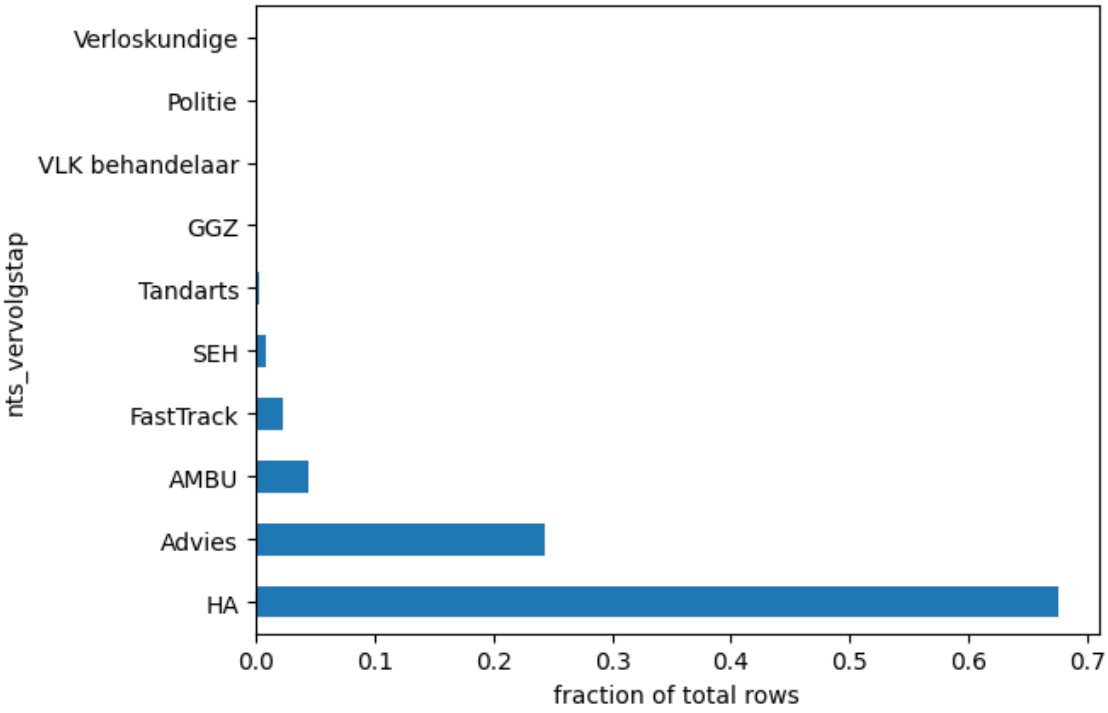


FIGURE 4.2: Occurrence of each the possible nts_vervolgstap as fraction of total rows

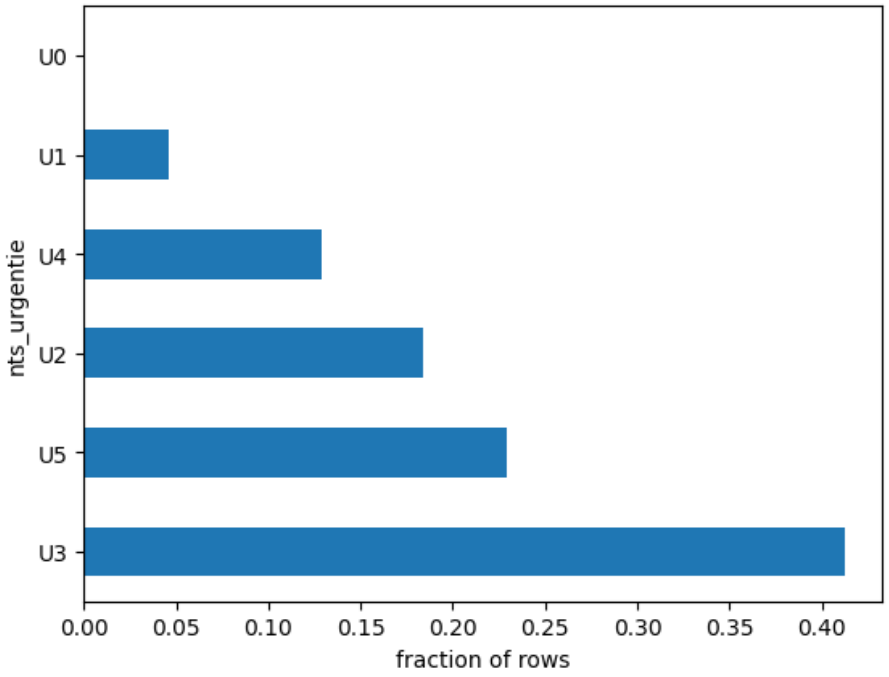


FIGURE 4.3: Occurrence of each possible `nts_urgentie` as fraction of total rows

at the start of a call during that time.

- Sometimes the triagist wrote an advice in the `algemeen` field which is part of `sda_regel` instead of in `aanvullend_advies`.
- The start of `p_regel` is often automatically filled with the `aanvullend_advies` and `gekozen_advies` in case those fields have values.
- The urgency level is sometimes also mentioned in `algemeen`
- First names or abbreviations like "mw."(ms) or "mr." are mentioned to refer to the caller in unstructured text fields.

4.4 Data cleaning

The dataset underwent a series of cleaning procedures as outlined below:

- Some rows were corrupted, where the id was a string of another column, these rows were removed.
- When a field of table 4.3 is not applicable, triagists use different methods to indicate this. Some leave the field empty while others for example type "geen", which means none. We account for this by converting these different cases to an empty string.
- `gekozen_adviezen` included HTML elements of the web application. These elements were removed using regular expressions.
- Given our focus on generating the unstructured text advice of the triagist, we drop rows where this field is missing.

- Although, most fields are not optional according to the NTS [1], the application allows the user to skip the urgency and next step, but the user has to give a reason for skipping. An example is when the person calling is asking for certain medication. We exclude these cases where rows are missing the urgency level or next step.
- We exclude rows where the length of `aanvullend_advies` is less than 20 characters. These advices are often vague and rarely contain useful information.

4.5 Input Data

The input data to our models consists of the following fields:

- All free text fields from table 4.3
- `ingangsklachten` from table 4.1
- `vragen` and `antwoorden` from table 4.1
- `nts_urgentie` and `nts_urgentie_gekozen` from table 4.1. We include both of these fields, because it shows whether a triagist changed the urgency, which could be valuable information for the model.
- `nts_vervolgstap` and `nts_vervolgstap_gekozen` from table 4.1. We include both of these fields, because it shows whether a triagist changed the next step, which could be valuable information for the model.
- The day of the week (Monday - Sunday), which is extracted from `begintijd` from table 4.1. This can be valuable information, because triagists may give different advice on certain days (for example during the weekend versus a weekday).

The values of the `aanvullend_advies` field will form the labels of our input data.

Chapter 5

Methods

This chapter goes over the proposed methodologies. First, we formalize our conditional language model. Next, we go over the models that were selected and how we adapted these to our use case. Then we will go over two fusion approaches that are used to combine unstructured text and categorical features. Next, we define the decoding strategies that were used. Furthermore, we define the evaluation metrics that are used to compare models. Finally, we define our method for adding explainability to our model outputs, using integrated gradients

5.1 Conditional language model for triage advice

Our problem can be modeled using a conditional language model that specifies a probability distribution $P(w_1, w_2, ..., w_n|c)$ over a sequence of n words $w_1, w_2, ..., w_n$ given a context c as also discussed in section 2.1. In our case, the unstructured text advice A is the sequence to be predicted. The context C is the input data outlined in section 4.5 and consists of the entry symptoms and their course SC, the question Q, the history H, medication M, drugs/alcohol D, selected entry symptoms S, questions and answers QA, proposed urgency PU, chosen urgency CU, proposed next step PS and chosen next step CS and the day of the week W. So, we formally define our language model as:

$$P(A|C = (SC, Q, H, M, D, S, QA, PU, CU, PS, CS, W)) \tag{5.1}$$

We train our language models using the cross-entropy loss. So, when generating an advice A consisting of T words given some context C, the training objective is to minimize the per word negative log-likelihood:

$$L(\theta) = -\frac{1}{T}\sum_{t=1}^{T} logP_\theta(w_t|w_1, w_2, ..., w_{t-1}, C) \tag{5.2}$$

We use teacher forcing [59] during training, which should lead to more stable training and faster convergence.

5.2 Model selection and adaptation

To answer ***SQ1*** we require two PLMs, one which is healthcare-specific and another which is domain-generic. We also require similar underlying architectures for both PLMs for a

fair comparison. Also, we are restricted to models that are pre-trained on Dutch text, since the data is Dutch. The only Dutch, healthcare-specific model is MedRoBERTa.nl [55], which is a RoBERTa-based [38] model. Given this limitation, we compare this model to a Dutch, domain-generic RoBERTa-based model called RobBERT [15]. However, these RoBERTa-based models consist of twelve encoder blocks and are encoder-only, which means they are not traditionally used for NLG. Therefore, we adopt the approach from Rothe et al. [45], which we discussed in section 3.1. MedRoBERTa.nl and RobBERT will both function as encoder and decoder, such that we have two encoder-decoder models that we will call Med2Med and Rob2Rob respectively. An overview of these models is given in table 5.1.

TABLE 5.1: Models that will be used in the experiments

Name	Encoder	Decoder	Pre-training
Med2Med	MedRoBERTa.nl	MedRoBERTa.nl	Healthcare-specific
Rob2Rob	RobBERT	RobBERT	Domain-generic

The architecture for Rob2Rob is depicted in figure 5.1. In the RobBERT decoder we mask the self-attention mechanism, such that the right context is hidden, i.e. the decoder can not look into the future when generating outputs. Furthermore, we add an extra attention block to the RobBERT decoder, such that we create a connection between the RobBERT encoder and decoder. The weights of this new attention block are randomly initialized because they are introduced by us and are therefore not pre-trained. The other attention blocks utilize the pre-trained weights from the model checkpoints. With these modifications, the architecture corresponds to the vanilla transformer discussed in section 2.2.4. The architecture of Med2Med is exactly the same. Only pre-training is different. This way we can see if domain-specific pre-training is useful for our triage situation.

The proposed architectures expect the input to be tokenized. RobBERT and MedRoBERTa.nl have accompanying tokenizers that have also already been trained. Therefore we also use these tokenizers for tokenizing the text input of Rob2Rob and Med2Med. Both tokenizers use byte-level Byte Pair Encoding(BPE). The vocabularies of the RobBERT and MedRoBERTa.nl tokenizers have different sizes. RobBERT has 40000 possible tokens while MedRoBERTa.nl has 52000 tokens. The separator token of the tokenizers is used to separate different input features. In case of RobBERT and MedRoBERTa.nl this is the </s> token. We considered introducing new special tokens into the tokenizer to separate each feature, which can potentially make it easier for the model to distinguish between different features. However, this increases the vocabulary of the tokenizer and thereby also increases the model size. Also, the model may be able to distinguish between the features based on the separator token and positional encoding. Therefore, we chose to use the already present separator token.

5.3 Fusion approaches

Our input consists of two modalities: unstructured text and categorical data. However, our architectures from 5.2 are traditionally designed to handle sequences. Specifically, they have been pre-trained on free-form text. Therefore, we propose two fusion approaches to incorporate categorical variables into the input.

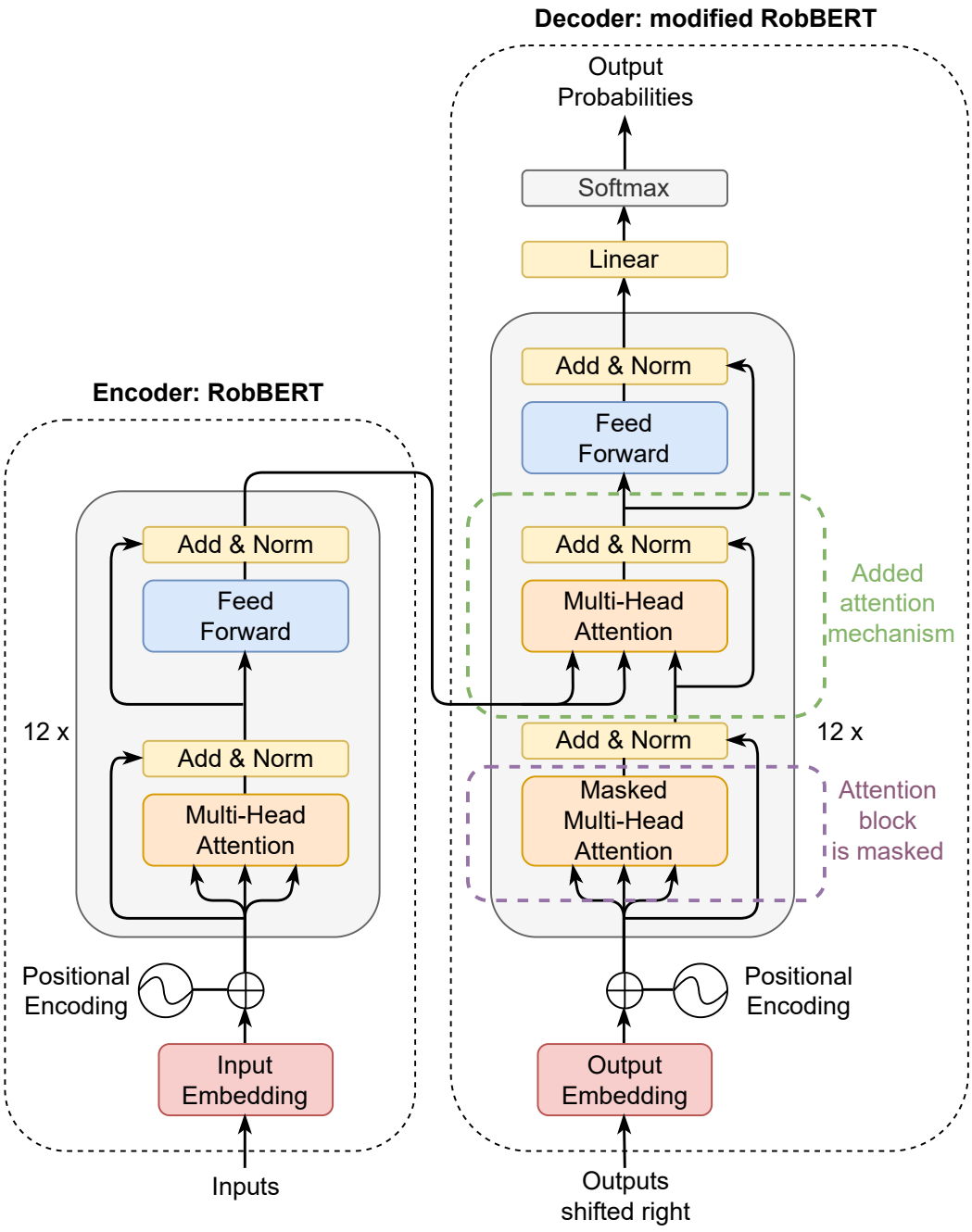


FIGURE 5.1: Rob2Rob architecture

5.3.1 All-Text

In this approach unstructured text fields and categorical fields are all concatenated into a single string using a separator token from the tokenizer as in listing 5.1.

Listing 5.1: All-Text input structure

```
<klacht_beloop></s><hulpvraag></s><voorgeschiedenis>
</s><medicatie></s><alcohol_drugs></s><ingangsklachten>
</s><vragen, antwoorden></s><nts_urgentie>
</s><nts_urgentie_gekozen></s><nts_vervolgstap>
</s><nts_vervolgstap_gekozen></s><day_of_week>
```

The categorical variable `ingangsklachten` can have multiple values, so if we concatenate this feature we separate each value using a comma. `vragen` and `antwoorden` may also have multiple values and a value itself can contain a comma, therefore we separate these values by a semicolon. An example input is shown in listing 5.2. This input can then be fed through the architecture depicted in 5.1. The benefit of this approach is its simplicity. The architecture of figure 5.1 does not have to be extended, which means we do not have to train extra parameters. Therefore, we require less resources. We also hypothesize that language models are able to handle the categorical features, given that they are text. A feature such as the urgency (`nts_urgentie_gekozen`) is a scale, therefore the model may have more difficulty to make sense of such a feature when it is concatenated as text, compared to for example `vragen` and `antwoorden` which are simple questions and answers. However, we simply concatenate each value of the categorical features without any modification.

Listing 5.2: All-Text example input

```
krijgt geen ontlasting sinds 3 dagen, buik is gespannen.
maakt zich zorgen. wat gevoelig aan linkerkant van
onderbuik.</s>kan ik mijn ontlasting kwijtraken dmv een
middeltje?</s>urineretentie, heeft momenteel catheter.
consult uroloog ivm dd vergrote
prostaattamsulosine</s>Obstipatie</s>
Braken: Nee; Buikpijn: Nee/nauwelijks (<4)</s>
U5</s>U5</s>Advies</s>Advies</s>Dinsdag
```

5.3.2 Fuse-Early

This approach is based on the Fuse-Early approach of Shi et al. [47]. In this approach, the text features are embedded using an encoder. Let d be the embedding dimension of the encoder and x our sequence of input tokens. For an input consisting of n tokens after tokenization, the encoder produces a hidden vector h_{x_i} of length d for each input token x_i. Together these hidden vectors form a matrix $X \in \mathbb{R}^{n \times d}$.

Let c be our categorical features that are embedded separately from the text features. For each categorical feature c_i, a vocabulary is created that maps each possible value to a distinct integer. So, if c_i has a cardinality of k. Each possible value of c_i will map to a discrete value between 1 and k, while 0 is set as the padding token, so that the embedding layers can handle batches.

After mapping a categorical variable c_i to an integer using its vocabulary, we use entity embeddings [22] to map each categorical feature c_i to a continuous vector with a lower embedding dimension e_i. A potential benefit of this approach over for example

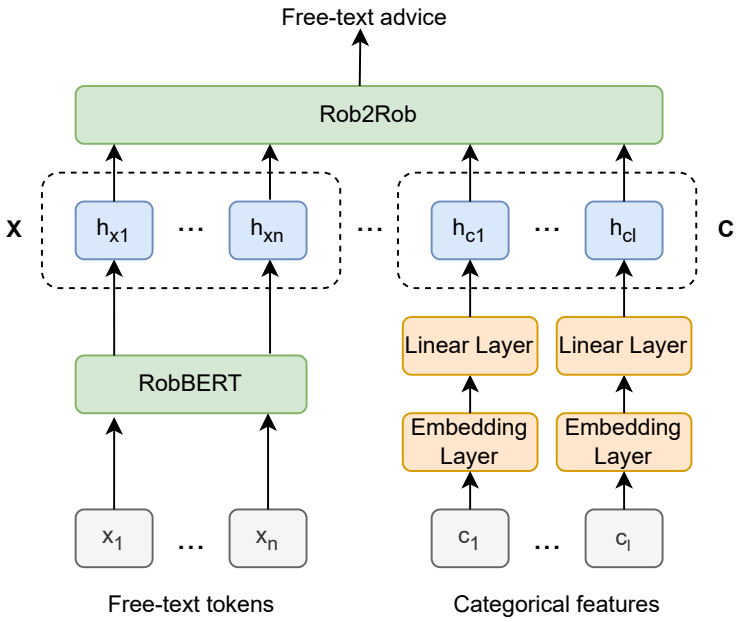


FIGURE 5.2: Fuse-Early architecture using RobBERT

one-hot-encoding, is that similar values are mapped close to each other in the embedding space, which can reveal intrinsic properties of the categorical variable c_i.

The embeddings of the unstructured text features and categorical features together serve as input to an encoder-decoder that is responsible for generating the unstructured text advice. However, this encoder-decoder has the same hidden dimension d as the encoder we use to embed the unstructured text features. Therefore, we transform the categorical features c to this dimension d using a single linear layer for each feature c_i, such that we get a hidden vector h_{c_i} for each feature c_i. Together these hidden vectors form a matrix $C \in \mathbb{R}^{l \times d}$. All hidden vectors from X and C are concatenated such that we have a hidden state $H \in \mathbb{R}^{(n+l) \times d}$, which forms the joint embedding of our unstructured text and categorical features:

$$H = \begin{bmatrix} X \\ C \end{bmatrix} = \begin{bmatrix} h_{x_1} \\ h_{x_2} \\ . \\ h_{x_n} \\ h_{c_1} \\ h_{c_2} \\ . \\ h_{c_l} \end{bmatrix} \tag{5.3}$$

As any other layer in the architecture, the embedding layers can update their weights through backpropagation and can therefore be trained. Figure 5.2 shows what this architecture looks like when we use RobBERT as the encoder for text-features and Rob2Rob as the encoder-decoder for generating text.

The downside of this approach, compared to All-Text, is the added complexity, which increases the size of the model in terms of parameters. The potential upside of this approach is that this model may be more capable of learning the interactions between the modalities.

5.4 Decoding methods

The advice written by triagists is concise and straight to the point, and therefore we do not consider sampling strategies, such as Top-k [18] and Top-p [24], as they are designed to generate more diverse and fluent outputs. However, we do use two other decoding strategies that we define in this section, which are a repetition penalty [31] and beam search.

5.4.1 Repetition penalty

Language models have been found to generate consecutive repetitions of n-grams, when generating an output [24, 62]. Keskar et al. [31] introduced a repetition penalty to mitigate this problem. This is done by discounting previously generated tokens, when computing the output probabilities for the next token.

Given that we have a score $x_i \in \mathbb{R}^d$ assigned to every token i within the tokenizer's vocabulary, the probability of selecting the ith token during the generation of an output using a language model, can be described as follows:

$$p_i = \frac{\exp(x_i)}{\sum_j \exp(x_j)} \tag{5.4}$$

The repetition penalty is then introduced as follows. If g are the previously generated tokens, then using the notation from equation 5.4 we have that the probability of outputting the ith token is given by:

$$p_i = \frac{\exp(x_i / I(i \in g))}{\sum_j \exp(x_j / I(j \in g))} \tag{5.5}$$

Where:

$$I(c) = \begin{cases} \theta, & \text{if } c = \text{True} \\ 1, & \text{otherwise} \end{cases} \tag{5.6}$$

5.4.2 Beam search

Beam search is a heuristic search algorithm that is used to find the most likely sequence of elements given a sequence generation model. It can be used to improve the quality of generated sequences by considering multiple alternative candidates at each step, as opposed to selecting output tokens greedily. It works as follows:

- We have a sequence-to-sequence model $P(y_t|y_{1:t-1})$, where y_t is the token at time step t and $y_{1:t-1}$ represents the sequence up to time step $t-1$.
- Let B be our beam with width k. k determines the number of candidate sequences to retain at each decoding step.
- B is initialized with empty sequences: B $= \{y_{1:0}\}$.
- For each time step t, where $t = 1, 2, ..., T$ and T is the maximum sequence length:
  - For each candidate sequence y in the current beam:
    1. Generate the set of next-token candidates Y_t by considering all possible tokens y_t and their associated probabilities $P(y_t|y_{1:t-1})$.

2. Calculate a score for each candidate sequence $y + y_t$ based on the product of its previous score and $P(y_t|y_{1:t-1})$:

$$s(y + y_t) = s(y) \cdot P(y_t|y_{1:t-1})$$

– Select the top-k sequences with the highest scores and add them to the beam:

$$B = \text{TopK}(\{y + y_t | \forall y \in B, \forall y_t \in Y_t\}, k)$$

- The search terminates when we have reached the maximum generation length of the model or when all candidate sequences have reached an end-of-sequence token.
- The output is the candidate sequence with the highest overall score s.

This way it balances exploration and exploitation by considering multiple alternative candidates at each decoding step, rather than selecting output tokens greedily. By maintaining a beam of the top candidate sequences, it can improve the quality of generated sequences.

5.5 Evaluation metrics

Evaluation metrics in NLG have mostly been designed for machine translation and summarization tasks. However, these metrics can still be applied to other tasks, since they measure the similarity between the model outputs and references. There is no standard metric for our task. Therefore, we use several commonly used metrics in NLG. Namely, ROUGE-1/ROUGE-2 F1 [36], BLEU [40] and METEOR [5].

ROUGE-N measures the overlap of n-grams between the generated text and the reference text. ROUGE-N P and ROUGE-N R are defined as follows. Where P is precision and R is recall.

$$\text{ROUGE-N P} = \frac{count(\text{matching n-grams})}{count(\text{n-grams in generated text})} \tag{5.7}$$

$$\text{ROUGE-N R} = \frac{count(\text{matching n-grams})}{count(\text{n-grams in reference text})} \tag{5.8}$$

The F1 score is then defined as follows.

$$\text{ROUGE-N F1} = \frac{2 * (\text{ROUGE-N P} * \text{ROUGE-N R})}{(\text{ROUGE-N P} + \text{ROUGE-N R}} \tag{5.9}$$

Similar to ROUGE-N, BLEU measures how many n-grams in the machine-generated text overlap with those in the reference text. BLEU takes into account precision and brevity. Brevity measures how close the length of the generated text is to the length of the reference text. It is calculated as follows.

$$\text{BLEU} = \text{BP} \cdot \exp\left(\sum_{n=1}^{N} w_n \cdot \log(p_n)\right) \tag{5.10}$$

Where N is the maximum n-gram order considered. w_n are the weights for each n-gram order. p_n is the precision for n-grams, calculated as the ratio of matching n-grams in the generated text to the total number of n-grams in the generated text. BP is the brevity

penalty, which is defined as follows. Where c is the length of the generated text and r is the length of the reference text.

$$\text{BP} = \begin{cases} \exp\left(1 - \frac{r}{c}\right) & \text{if } c \leq r \end{cases} \tag{5.11}$$

METEOR was designed to address some limitations of BLEU and is calculated as follows.

$$\text{METEOR} = (1 - \alpha) \cdot p + \alpha \cdot r \cdot f_\beta \tag{5.12}$$

Where α, balances precision and recall. p is the precision, which is defined as the ratio of matching unigrams in the generated text to the total number of unigrams in the generated text. r is the recall, which is defined as the ratio of matching unigrams in the generated text to the total number of unigrams in the reference text. f_β is the F-beta score, which combines p and r and is defined as follows.

$$f_\beta = \frac{(1 + \beta^2) \cdot p \cdot r}{\beta^2 \cdot p + r} \tag{5.13}$$

5.6 Model explainability

To aid triagists in decision making when using our model, we aim to add explainability to our model outputs. As discussed in 3.5, attribution methods can be used to explain the model outputs of deep networks. In NLP these methods are used to determine which input had a high attribution to the output of the model. In an NLG task, a model produces multiple output tokens rather than a single output. When generating a token, the model takes into account the input tokens and previously generated tokens. Therefore, attributions of input tokens and previously generated tokens are calculated for each output token individually. We use integrated gradients (IG) [51], because this method does not violate the fundamental axioms for attribution methods, as explained in section 3.5.

5.6.1 Integrated Gradients

Our method for using integrated gradients on a sequence-to-sequence task follows the approach proposed by Alammar [2]. Using an encoder-decoder model we autoregressively generate an output sequence y based on an input sequence x of and previously generated tokens p. Let l be the number of output tokens in y, n the number of tokens in x and z the concatenation of x and p. Then for every output token y_t, where $\forall t \in \{1, .., l\}$ (y_0 represents the encoder-decoder start token), we calculate the integrated gradient IG_{it} for every token z_i, where $\forall i \in \{0, .., n + t - 1\}$

The following procedure is followed to calculate IG_{it}. Let z' be our baseline, which has the same shape as z, but instead is solely filled with the padding token of the encoder-decoder. Integrated gradients can be described as the integral of gradients along a direct line from baseline z' to z. To approximate this, a straight line path is interpolated from z' to z using m number of steps. For every step interval j the gradient is computed. The integrated gradients can then be approximated by accumulating these gradients. The integrated gradient for a specific z_i when generating an output token y_t can then be retrieved as follows. Here $\frac{\partial F}{\partial z_i}$ is the gradient of our encoder-decoder F along the i^{th} dimension.

$$IG_{it}(F, z, z') = (z_i - z'_i) \sum_{j=0}^{m} \frac{\partial F(z' + \frac{j}{m}(z - z'))}{\partial z_i} \frac{1}{m} \tag{5.14}$$

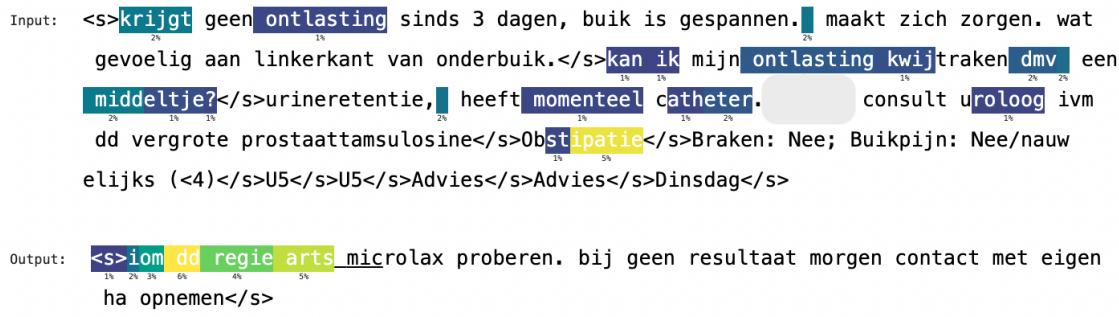


FIGURE 5.3: Example of a visualization that shows the Integrated Gradients attributions of the input tokens and previously generated tokens for the generated token " mic". Attributions are only shown if they are above 1%. It is created using D3 [9]. A date that was present in the data has been hidden.

IG_{it} is a vector with a length equal to the embedding dimension d of the encoder-decoder. It represents the integrated gradient of token z_i when generating y_t. We calculate the L2 norm for each of these vectors as follows.

$$\|IG_{it}\|_2 = \sqrt{\sum_{j=1}^{d} |IG_{itj}|^2} \tag{5.15}$$

Let a be the number of elements in z. Then we normalize each element $\|IG_{it}\|_2$ such that we get an attribution score s_i for each token z_i as follows.

$$s_{z_i t} = \frac{\|IG_{it}\|_2}{\sum_{j=1}^{a} \|IG_{itj}\|_2} \tag{5.16}$$

Repeating this procedure for each output token, results in an attribution score for each input token and previously generated token for all tokens in the output y.

5.6.2 Aggregating attributions

The visualization approach presented in section 5.6.1 and figure 5.3 is too complex for our end-user, since they would have to hover over different output tokens. This is not desirable, since a triage is conducted quickly. So, having to analyse the output in this way would take too much time. So, to simplify it further, we aim to aggregate the attribution scores to retrieve attribution scores of the input tokens for the entire output sequence as a whole.

For this aggregation we only consider the input tokens x, since we aim to show which input words attributed most to the proposed output of the model, thereby giving some form of explanation. So, the procedure for calculating integrated gradients is the same as in the previous subsection, but after calculating all the attribution vectors IG_{it}, we ignore the attributions of the previously generated tokens which were denoted as p in the previous subsection. IG_{it} is a vector with a length equal to the embedding dimension d of the encoder-decoder, representing the attribution of x_i in generating y_t. We will call this vector v_{it}. So for each output token y_t, we retrieve a vector v_{it} for each input token x_i. For each x_i, we aggregate these attributions into a single vector v_i as follows. Where l is the number of tokens in the output y.

$$v_i = \sum_{t=0}^{l} v_{it} \tag{5.17}$$

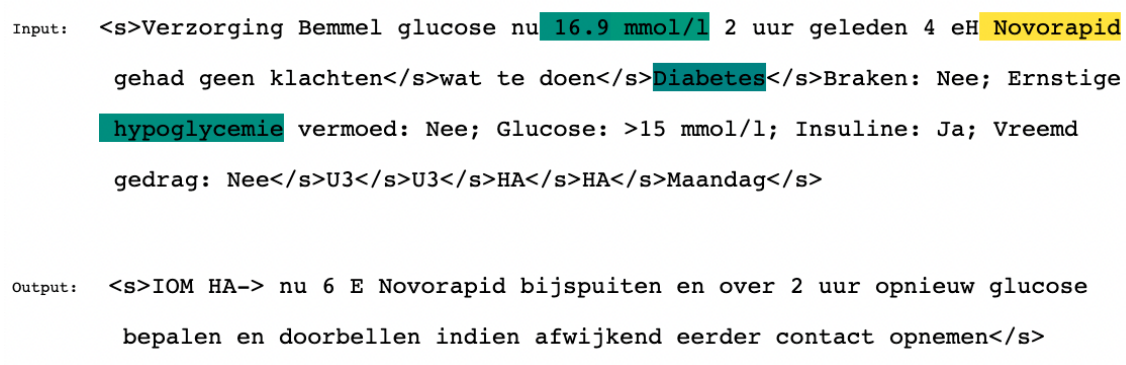


FIGURE 5.4: Example visualization that shows aggregated Integrated Gradients attributions for the input words. Attributions are only shown if they are above 5%. It is created using D3 [9].

We then use the L2 norm, such that we have a single score for each input token x_i for the entire output sequence y.

$$\|v_i\|_2 = \sqrt{\sum_{j=1}^{d} |v_{ij}|^2} \tag{5.18}$$

Then we normalize each element $\|IG_{it}\|_2$ such that we get an attribution score s_i for each token input token x_i for the entire output sequence y as follows. Where n is the number of tokens in x.

$$s_{x_i} = \frac{\|v_i\|_2}{\sum_{j=1}^{n} \|v_{ij}\|_2} \tag{5.19}$$

To make the output more understandable for the end-user we show input word attributions instead of input token attributions. This is done by simply adding the scores of input tokens in x if they together form a word. The attributions can then be visualized, using for example a color scale, as shown in figure 5.4. It indicates to the user which words the model deemed as important when generating the output. We test the usefulness of this approach during a user study which is presented in chapter 7.

Chapter 6

Experiments

6.1 Experimental setup

The dataset that results from the cleaning steps outlined in section 4.4 consists of 120751 rows. This set is randomly split using a random seed into a train, validation and test. We use a 80%/10%/10% split as shown in table 6.1. We use the same seed across experiments, such that the split remains the same.

Split	Fraction	Entries
Train	0.8	96600
Validation	0.1	12075
Test	0.1	12076

TABLE 6.1: Dataset split

In order to compare different models we use a training setup that is almost identical across experiments. We use a single NVIDIA T4 with 16GB of memory. We use Adam [33] as the optimiser. The learning rate is fixed at 5e-5, the epsilon at 1e-8 and weight decay at 0.01. A batch size of 4 or 8 is used, depending on the model size. Cross-entropy loss is used as the loss function for all models. The loss on the validation set is determined after each epoch. All models are trained for a variable number of epochs until the validation loss starts to increase. PyTorch [41] is used to implement all models.

6.2 Evaluation

We use ROUGE-1/ROUGE-2 F1 [36], BLEU [40] and METEOR [5] and to compare different models. All metrics have been loaded from the `evaluate` library of Huggingface [60] and we keep all parameters at default. The metrics were unanimous in terms of performance when comparing different models. Therefore, we report the ROUGE-1 and ROUGE-2 F1 score in this chapter, which has also been previously used to score NLG models in a healthcare setting [37]. All full overview of metric scores for the first 4 experiments can be found in appendix A.

6.3 Experiment 1: Rob2Rob vs Med2Med (*SQ1*)

To answer ***SQ1***, which aims to determine whether healthcare-specific pre-training is beneficial over domain-generic pre-training in our situation, we train Rob2Rob and Med2Med

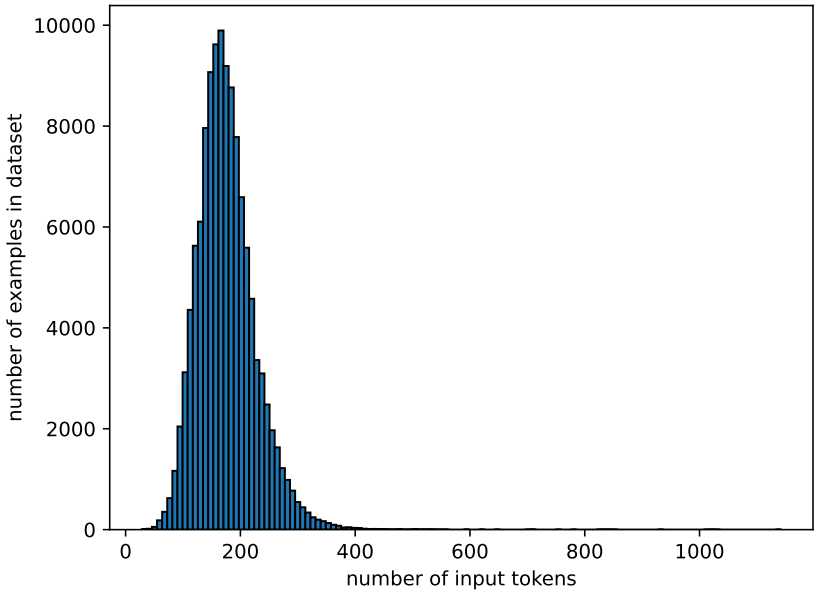


FIGURE 6.1: Histogram with 125 bins of the number of input tokens after tokenization of the whole dataset in the All-Text approach with categorical features

on our dataset.

We use the `transformers` library from Huggingface [60] to set up Rob2Rob and Med2Med using the RobBERT v2 base [15] and MedRoBERTa.nl [55] checkpoints, respectively. Apart from the difference in checkpoints, the setup of Rob2Rob and Med2Med and their tokenizers are identical. Therefore, we only outline the setup procedure for Rob2Rob.

Most of the weights of Rob2Rob, which is depicted in 5.1, are initialized using the RobBERT v2 base checkpoint. Only the weights of the cross-attention block are randomly initialized, since this block is newly introduced into the architecture and can therefore not leverage the checkpoint. So, these cross-attention weights are learned from scratch. Rothe et al. [45] experimented with sharing the weights between the encoder and decoder. They found that it only leads to small drops in performance and sometimes even slightly increases performance, while reducing the memory footprint. Therefore, we chose to also share the weights between the encoder and decoder. The tokenizer of Rob2Rob uses the default special tokens from the RoBERTa [38] tokenizer. This includes for example the padding token `<pad>`. The vocabulary of the tokenizer is loaded from the model RobBERT v2 base checkpoint.

We use the All-Text approach from section 5.3.1 to prepare the inputs for the tokenizers. To determine the maximum number of tokens of our inputs, we tokenized the entire dataset (so before train/validation/test split) and determined the length of all input sequences after tokenization, as depicted in 6.1. We see that the distribution has a long tail with outliers. We are forced to truncate all inputs at least to a length of 512, as this is the maximum number of tokens that our PLMs can process per input. However, we took this a step further and truncated any sequence that fell into the top 0.05% of lengths, which is a sequence length of 386 tokens. So, we set the `max_length` of the tokenizer to 386 and `truncation` to `True`. This reduces the memory overhead, because we require less padding of batches. All other tokenizer parameters were kept at default.

The models were trained with a batch size of 8, until the validation loss increased. This was after 4 epochs for both models. Therefore, we evaluate both models on the test set after epoch 3.

The ROUGE-1 and ROUGE-2 scores of both models are given in table 6.3. We observe that Rob2Rob slightly outperforms Med2Med when using the All-Text approach with all features. This shows that healthcare-specific pre-training on hospital notes is not beneficial compared to domain-generic pre-training in this case. This is in line with previous work, where MedRoBERTa.nl and RobBERT showed similar performance in predicting lung cancer [17] and classifying unstructured reports on their diagnostic goal [44]. There can be different explanations for this.

First of all, MedRoBERTa.nl was pre-trained on Dutch hospital notes with a total size of roughly 25.8GB, while RobBERT was trained on a Dutch corpus that was 39GB in size. So, RobBERT's pre-training was more extensive, which may have put Rob2Rob in a better initial state compared to Med2Med.

Secondly, people usually call an out-of-hours primary care center in case of minor problems and therefore they do not undergo an assessment by a doctor. Consequently, the amount of medical terms used in the texts is likely limited compared to the hospital notes that were used to train MedRoBERTa.nl. For instance, in the MedRoBERTa.nl paper [55] they give the example text: *"Patiënt met possibele pulmonale aspergillus met oplopend galactomannan onder vori mono"*, which translates to *"Patient with possible pulmonary aspergillus with ascending galactomannan under vori mono"*. We rarely see texts with so many Latin/Greek medical terms in our data. In our data the texts are usually more straightforward such as: *"Heeft een blaasontsteking, donderdag klachten begonnen. Branderig gevoel bij het plassen en vaak plassen, plast ook kleine beetjes. Geen temp. Drinkt slecht omdat zij dan bang is te moeten plassen."*, which translates to: *"Has a bladder infection, Thursday symptoms started. Burning sensation when urinating and urinates often, also urinates small bits. No temp. Drinks poorly because she is then afraid to urinate."*. As such, Med2Med may not be able to fully leverage the healthcare-specific pre-training on hospital notes.

In addition to a slightly worse performance, Med2Med is roughly 6% larger in size compared to Rob2Rob in terms of trainable parameters. However, since we used an identical batch size, the extra training time was only about 2% in our experiment. The number of parameters of Rob2Rob and Med2Med and the training time per epoch for this experiment, are given in table 6.2. Given the slightly better performance and smaller model size, we use Rob2Rob in subsequent experiments.

TABLE 6.2: Number of trainable parameters of Med2Med and Rob2Rob and the training time per epoch for experiment 1.

Model	**number of parameters (million)**	**Training time (min/epoch)**
Med2Med	≈ 146	≈ 102
Rob2Rob	≈ 155	≈ 105

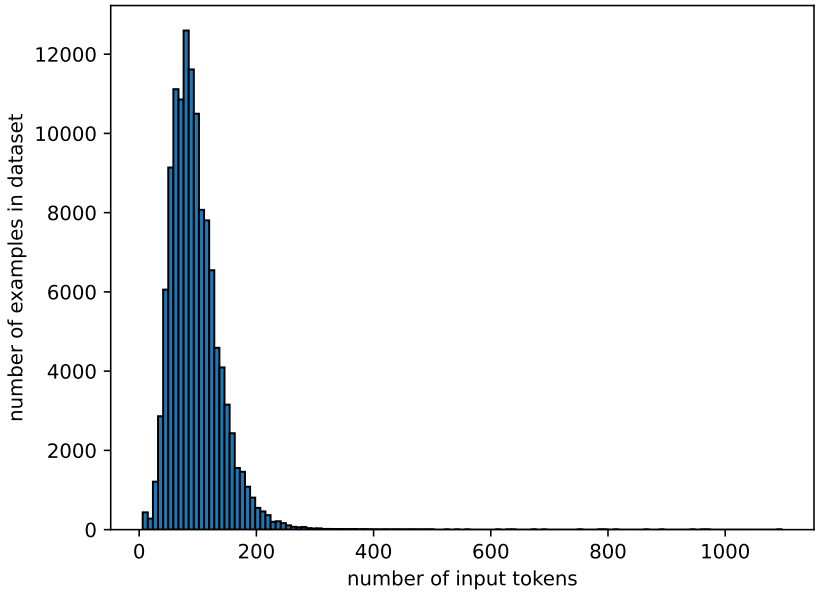


FIGURE 6.2: Histogram with 125 bins of the number of input tokens after tokenization of the whole dataset in the All-Text approach without categorical features

6.4 Experiment 2: Rob2Rob free-form textual features only (*SQ2*)

The goal of this experiment is to answer ***SQ2***, which aims to determine whether the categorical input fields have added value when generating a triage advice.

In experiment 1 we already trained Rob2Rob using an All-Text approach. Here we adopt the same approach, but we only concatenate the unstructured text features. These fields consist of the following: entry symptoms and their course, the question, the history, medication and drugs/alcohol. So, the features that were originally categorical are not included. The structure of this input is shown in listing 6.1.

LISTING 6.1: All-Text input with only unstructured text features

```
<klacht_beloop></s><hulpvraag></s><voorgeschiedenis>
</s><medicatie></s><alcohol_drugs>
```

The model was trained with a batch size 8, until the validation loss increased. This was after 5 epochs. Therefore, we evaluate the model after 4 epochs on the test set.

The ROUGE-1 and ROUGE-2 scores of the trained model is given in table 6.3. We observe that the performance of Rob2Rob decreases when only unstructured text features are included in the All-Text approach. This indicates that the categorical features contain information that is valuable to the model when it generates an advice. Therefore, it is interesting to explore a different fusion strategy for these features.

Model	Fusion approach	Only unstructured text features	encoder/decoder weights shared	ROUGE-1 F1	ROUGE-2 F1
Rob2Rob	All-Text	No	Yes	27.5	12.0
Med2Med	All-Text	No	Yes	27.2	11.8
Rob2Rob	All-Text	Yes	Yes	22.0	8.1
Rob2Rob	Fuse-Early	No	Yes	18.4	3.6
Rob2Rob	All-Text	No	No	27.9	12.4

TABLE 6.3: ROUGE-1 and ROUGE-2 F1 scores on the test set, for all models that were trained in experiments 1-4.

6.5 Experiment 3: All-Text vs Fuse-Early (*SQ2*)

The results of experiment 2 show that, when the categorical variables are concatenated as text, they contain useful information that Rob2Rob can use when generating an advice. Therefore, it is interesting to explore a more complex fusion approach. So, to answer ***SQ3***, we explore the Fuse-Early approach from section 5.3.2 with Rob2Rob.

The archictecture of this approach is depicted in figure 5.2. We chose to not have separate embedding layers for `vragen` and `antwoorden`. `vragen` are questions a triagist has to ask based on `ingangsklachten`. There are 306 unique questions in total and for a particular input only a handful will be relevant, therefore we deem it wasteful to have a separate embedding layer for each question. So, we chose to concatenate `vragen` and `antwoorden` as text as in the All-Text approach.

The text encoder RobBERT is loaded from the RobBERT v2 base [15] checkpoint without any modification. The tokenizer of RobBERT is also loaded from the same checkpoint and is identical to the tokenizers of experiment 1 and 2. However, we again truncated any text input that fell in the top 0.05% of lengths. This is a sequence length of 369 tokens. So, we set the `max_length` of the tokenizer to 369 and `truncation` to `True`. All other tokenizer parameters were kept at default.

The setup for Rob2Rob is the same as in experiments 1 and 2. Rob2Rob does not require a tokenizer in this setup, since the inputs are already embedded by the RobBERT encoder and the embedding layers for categorical features as described in section 5.3.2.

Each remaining categorical feature is embedded using a separate embedding layer as shown in figure 5.2. The configurations of these layers are shown in table 6.4.

Categorical feature	Cardinality	Embedding dimension
`ingangsklachten`	61	32
`nts_urgentie`	5	4
`nts_urgentie_gekozen`	5	4
`nts_vervolgstap`	11	4
`nts_vervolgstap_gekozen`	11	4
`dag`	7	6

TABLE 6.4: Embedding layer configuration for each categorical feature in the Fuse-Early approach of experiment 3

The model was trained with a batch size of 4 until the validation loss increased. This was after 6 epochs. Therefore, we evaluate the model after 5 epochs on the test set.

The ROUGE-1 and ROUGE-2 scores of the model trained with Fuse-Early approach are given in table 6.3. We observe that the performance of Rob2Rob decreases significantly when using the Fuse-Early approach. The model is even outperformed by the model that only considers unstructured text features.

The size of our dataset could be a limiting factor for the Fuse-Early approach. First of all, because there are more trainable parameters in general, compared to the All-Text approach. Secondly, more of these parameters, such as the new embedding layers, have to be trained from scratch.

Another potential problem with the Fuse-Early approach may lie in the fusion of the embeddings of the unstructured text and categorical features. Rob2Rob has an embedding layer that is used to get embedding vectors based on the input tokens that are obtained using its tokenizer. However, in our Fuse-Early approach we pass the input embeddings, which we construct ourselves by concatenating the unstructured text embeddings and categorical embeddings, directly to Rob2Rob. So, these categorical features do not pass through the embedding layer of Rob2Rob. Therefore, Rob2Rob may struggle to make sense of these embeddings in combination with the embeddings from the unstructured text features.

Finally, we used entity embeddings, which projects each categorical feature to a lower dimension. However, the hidden layer dimension of Rob2Rob is fixed at 768, which forced us to project each of those embeddings to that dimension before passing them to Rob2Rob. These projections may not be optimal, which could have caused a loss of information.

6.6 Experiment 4: Encoder-Decoder weight sharing (*SQ3*)

In all previous experiments we shared the weights between the encoder and decoder. However, this is based on the assumption that sharing these weights does not hurt performance [45]. To confirm this, in this experiment, we used an identical setup to experiment 1, which is the Rob2Rob model with an All-Text approach, but this time the weights are not shared between the encoder and decoder.

The model was trained with a batch size of 4 until the validation loss increased. This was after 4 epochs. Therefore, we evaluate the model after 3 epochs on the test set.

The ROUGE-1 and ROUGE-2 scores of Rob2Rob, trained with an All-Text approach and where the weights are not shared between the encoder and decoder, are given in table 6.3. We observe that it slightly outperforms the same Rob2Rob model that does share the weights between the encoder and decoder. The difference is minor, which is inline with the findings of Rothe et al. [45]. Not sharing the weights between the encoder and decoder, increases the number of trainable parameters by almost 80%, which forced us to decrease the batch size from 8 to 4. Overall, it led to an increase in training time of almost 24%. Given the small decrease in performance in terms of ROUGE scores and the large decrease in memory footprint, we conclude that weight-sharing is beneficial. Therefore, we keep using this in subsequent experiments.

6.7 Experiment 5: Preventing n-gram repetitions (*SQ3*)

During manual examination of the outputs of our Rob2Rob model that resulted from Experiment 1, we observed that consecutive uni- and bi-gram repetition occurred. For example, the following uni-gram repetition was found in one of the outputs of the test set:

Voor nu geen alarmsymptomen uitleg drukverband en verband. Advies om met verband verband verband te verbinden. Akkoord

The word "verband" is repeated multiple times consecutively, which is obviously unwanted behaviour. Especially, if we consider the goal of the model, which is to reduce the workload of triagists. Therefore, having to manually remove repeated words is undesirable. In the examples of the test set we found 34 examples with uni-gram repetition and 5 examples with bi-gram repetition. Given that there are 12076 in the test set, this is only a small portion. However, we still explore if this problem can be easily overcome.

In an attempt to tackle this problem, we introduce a repetition penalty when generating outputs. This is done as described in section 5.4.1. This penalty is set to 1.2 as suggested by Keskar et al. [31]. We repeat the evaluation of the All-Text Rob2Rob model that was already trained in Experiment 1. So, the model checkpoint after 3 epochs is evaluated again on the test set, but this time with the repetition penalty in place.

Using the repetition penalty resulted in a ROUGE-1 score of 27.8 and a ROUGE-2 score of 11.9. The original evaluation resulted in a ROUGE-1 of 27.5 and ROUGE-2 of 12.0 as shown in table 6.3. So, the penalty led to a slight increase in ROUGE-1 score and a slight decrease in ROUGE-2 score. Furthermore, only one output with uni-gram repetition was found in the outputs of the test set, showing that the penalty mitigates the problem.

6.8 Experiment 6: Beam search (*SQ3*)

In all of our previous experiments greedy search was used, where at every time step the language model outputs the token with the highest probability. As also explained in section 3.3, beam search can be used to increase performance, because it keeps track of the n most likely beams and finally choosing the beam with the highest overall probability. Beam search is not used during training, since we use teacher forcing there. Therefore, we repeat the evaluation of the All-Text Rob2Rob model that was already trained in Experiment 1. This comes down to evaluating the model checkpoint after 3 epochs on the test set, using different beam widths.

The results are given in table 6.5. Note that a beam width of 1 corresponds to greedy search. We can see that using beam search does indeed increase the performance of the model in terms of ROUGE scores compared to greedy search. However, interestingly enough the performance decreases again when the beam width is set to 4 or more. A possible explanation for this, which was found by Cohen et al. [12], is that when the beam width is increased, it results in sequences that primarily begin with early, highly improbable tokens, followed by a sequence of tokens with higher conditional probabilities. They showed that such sequences tend to receive lower evaluation scores compared to sequences with lower probabilities without that pattern.

6.9 Experiment 7: Adding more input context (*SQ2*)

During our analysis of the user study results of chapter 7, we found that our model sometimes focuses on irrelevant parts of the input. A part of the triage information of an example case where we observe this, is given in table 6.6. The medication field lists the caller's medication. We can see that the caller uses tramadol, but was calling with symptoms of hyperventilation. Our attribution method marked the word "tramadol" as import and the model gave the advice to use tramadol. However, the triagist that evaluated this case noted that it does not make sense to use tramadol in case of hyperventilation. It

Beam Width	ROUGE-1 F1	ROUGE-2 F1
1	27.5	12.0
2	28.1	12.6
3	28.2	12.7
4	28.0	12.7
6	27.7	12.6
8	27.4	12.4

TABLE 6.5: ROUGE-1 and ROUGE-2 F1 scores on the test set for Rob2Rob with the All-Text approach and weight sharing between the encoder and decoder, when using beam search with different beam widths.

is worth noting that the input for these cases was generated by simply combining all input features using a single separator token. This suggests that our model might benefit from more context. Therefore, we explore 2 different approaches to provide more context with regard to input features.

TABLE 6.6: Information of a case that was assessed by a domain expert. The 5 most important words according to our attribution method are marked green. Some input fields are not shown for brevity.

	Dutch	**Translated**
Symptoms	denkt hyperventilatie aanval te hebben, klachten al uur bezig. Moet woensdag naar cardioloog voor onderzoek.	Suspects hyperventilation attack, symptoms been going on for hours. Must see cardiologist on Wednesday for examination.
Question	wil graag even luisterend oor zodat aanval afneemt.	Would like a listening ear so attack subsides.
History	longklachten, alcohol misbruik, paniek aanvallen. dec. cordis AP klachten.	lung complaints, alcohol abuse, panic attacks. dec. cordis AP complaints.
Medication	pufjes, tramadol.	puffs, tramadol.
Entry symptoms	Kortademig	Short of breath
Generated advice	i.o.m dd ra mag mw tramadol gaan nemen. Bij toename klachten opnieuw contact HAP.	In consultation with the doctor the woman can take tramadol. If symptoms increase contact HAP again.

The first approach aims to leverage the natural language knowledge already present in the model by adding a prefix to each input feature. The prefixes that were used are given in table 6.7. So, we still concatenate all features using the separator token `</s>`, but we prefix each feature value. An example input is given in listing 6.2.

LISTING 6.2: Example All-Text input with feature prefixes. Prefixes are marked yellow.

```
De klachten zijn: baby van 10 maanden, van oudere
dochter 30 tum tum zonder kauwen doorgeslikt. Zonder
toezicht gebeurd, dochter zag dat hij ineens leeg was.
Dus waarschijnlijk doorgeslikt, loopt nu rond. Niet
benauwd, niet kortademig, geen kuchen, is lekker
fanatiek, hoor haar ook op de achtergrond vrolijk
brabbelen. Loopt wat te kwijlen, lijkt nu opeens wel
meer. 3 snoepjes ui de mond gevist, koker is leeg.
</s>De hulpvraag is: advies</s>De ingangsklachten zijn:
Corpus alienum</s>Dit zijn gestelde vragen en gegeven antwoorden:
Corpus alienum: Nee; Corpus alienum ingeslikt/ingeademd:
Ja, zonder klachten; Doorboord: Nee; Oogletsel: Nee;
Pijnlijk oog: Nee; Stoornis doorbloeding: Nee</s>
De voorgestelde urgentie is U5</s>De gekozen urgentie is U5</s>
De voorgestelde vervolgstap is Advies</s>De gekozen vervolgstap is
Advies</s>Het is vandaag Zaterdag
```

TABLE 6.7: Prefixes used to concatenate input features

Input feature	**Prefix**	**Prefix translation**
klacht_beloop	De klachten zijn:	The complaints are:
hulpvraag	De hulpvraag is:	The help question is:
voorgeschiedenis	De voorgeschiedenis is:	The history is:
medicatie	De volgende medicatie wordt gebruikt:	The following medication is already being used:
alcohol_drugs	Alcohol/drugs:	Alcohol/drugs:
ingangsklachten	De ingangsklachten zijn:	The entry symptoms are:
vragen_antwoorden	Dit zijn de gestelde vragen en gegeven antwoorden:	These are the questions asked and answers given:
nts_urgentie	De voorgestelde urgentie is	The proposed urgency is
nts_urgentie_gekozen	De gekozen urgentie is	The chosen urgency is
nts_vervolgstap	De voorgestelde vervolgstap is	The proposed next step is
nts_vervolgstap_gekozen	De gekozen vervolgstap is	The chosen next step is
dag	Het is vandaag	Today it is

The second approach introduces a new token into the tokenizer for each input feature. These tokens are used to prefix each feature. The tokenizer will have to learn all the weights for these new tokens from scratch. The idea is that the model will be able to be better distinguish between different features when those features are separated with distinct tokens. In principle these tokens can be any token that is not already present in the vocabulary of the tokenizer. The tokens that we introduced are given in table 6.8. When concatenating the features we do not use the </s> token anymore, since this seems redundant given that the new tokens already separate the features. An example input is given in listing 6.3

Listing 6.3: Example All-Text input with token prefixes newly introduced into the tokenizer

```
[KLACHT_BELOOP]baby van 10 maanden, van oudere dochter
30 tum tum zonder kauwen doorgeslikt. Zonder toezicht
gebeurd, dochter zag dat hij ineens leeg was. Dus
waarschijnlijk doorgeslikt, loopt nu rond. Niet benauwd,
niet kortademig, geen kuchen, is lekker fanatiek, hoor
haar ook op de achtergrond vrolijk brabbelen. Loopt wat te
kwijlen, lijkt nu opeens wel meer. 3 snoepjes ui de mond
gevist, koker is leeg.[HULPVRAAG]advies
[INGANGSKLACHTEN]Corpus alienum[VRAGEN_ANTWOORDEN]
Corpus alienum: Nee; Corpus alienum ingeslikt/ingeademd:
Ja, zonder klachten; Doorboord: Nee; Oogletsel: Nee;
Pijnlijk oog: Nee; Stoornis doorbloeding:
Nee[URGENTIE]U5[URGENTIE_GEKOZEN]U5
[VERVOLGSTAP]Advies[VERVOLGSTAP_GEKOZEN]Advies
[DAG]Zaterdag
```

Table 6.8: Tokens introduced into the tokenizer to use as prefix for each input feature.

Input feature	**Token**
klacht_beloop	[KLACHT_BELOOP]
hulpvraag	[HULPVRAAG]
voorgeschiedenis	[VOORGESCHIEDENIS]
medicatie	[MEDICATIE]
drugs_alcohol	[DRUGS_ALCOHOL]
ingangsklachten	[INGANGSKLACHTEN]
vragen_antwoorden	[VRAGEN_ANTWOORDEN]
nts_urgentie	[URGENTIE]
nts_urgentie_gekozen	[URGENTIE_GEKOZEN]
nts_vervolgstap	[VERVOLGSTAP]
nts_vervolgstap_gekozen	[VERVOLGSTAP_GEKOZEN]
dag	[DAG]

We also extended this approach from listing 6.3 further by extending the tokenizer with a new token for each possible value of our categorical variables. Similar to experiment 3 we exclude `vragen` and `antwoorden`, because there are 306 unique questions in total with different possible answers. Therefore, we think it is wasteful to have a separate token for each possible question and each possible answer. So, we chose to concatenate `vragen` and `antwoorden` as text as in the All-Text approach. For remaining categorical variables we add a new token to vocabulary of the tokenizer. This was done by taking each value and surround it by square brackets. So, for instance, the urgency value `U5` is represented as `[U5]` in the tokenizer. An example input of this extended approach is given in listing 6.4.

LISTING 6.4: Example All-Text input with token prefixes and categorical value tokens newly introduced into the tokenizer

```
[KLACHT_BELOOP]baby van 10 maanden, van oudere dochter 30
tum tum zonder kauwen doorgeslikt. Zonder toezicht gebeurd,
dochter zag dat hij ineens leeg was. Dus waarschijnlijk
doorgeslikt, loopt nu rond. Niet benauwd, niet kortademig,
geen kuchen, is lekker fanatiek, hoor haar ook op de
achtergrond vrolijk brabbelen. Loopt wat te kwijlen, lijkt
nu opeens wel meer. 3 snoepjes ui de mond gevist, koker is
leeg.[HULPVRAAG]advies[INGANGSKLACHTEN][Corpus alienum]
[VRAGEN_ANTWOORDEN]Corpus alienum: Nee; Corpus alienum
ingeslikt/ingeademd: Ja, zonder klachten; Doorboord: Nee;
Oogletsel: Nee; Pijnlijk oog: Nee; Stoornis doorbloeding:
Nee[URGENTIE][U5][URGENTIE_GEKOZEN][U5][VERVOLGSTAP][Advies]
[VERVOLGSTAP_GEKOZEN][Advies][DAG][Zaterdag]
```

So, we trained a Rob2Rob model with weight sharing, 3 times with the 3 different approaches described above. Each model was trained with a batch size of 8 until the validation loss increased. We evaluate each model on the test set during the epoch before the validation loss increases. During evaluation a beam width of 3 and a repetition penalty of 1.2 is used.

Table 6.9 shows the results of this experiment. As can be seen the performance of each model improved in terms of ROUGE score when compared to the original model that simply concatenated all features using a separator token `</s>`. The approach where natural language prefixes are introduced has the highest performance. Adding categorical value tokens to the tokenizer in addition to adding special token prefixes, only increased the performance by roughly 0.1%.

TABLE 6.9: ROUGE scores on the test set, with a beam width of 3 and a repetition penalty of 1.2, for different input processing approaches

Input approach	**With categorical value tokens**	**Epochs**	**ROUGE-1 F1**	**ROUGE-2 F1**
Concat with </s>	No	3	28.3	12.8
Natural language prefixes	No	5	29.3	13.6
Special token prefixes	No	4	29.1	13.3
Special token prefixes	Yes	5	29.2	13.5

If we look at the advice of the cases of our user study from chapter 7 that were assessed with "Strongly Disagree" and "Disagree" we can see some changes. For instance, the example from earlier where tramadol is given as advice for a case of hyperventilation:

> *i.o.m dd ra mag mw tramadol gaan nemen. Bij toename klachten opnieuw contact HAP.*

Resulted in different advice for all three approaches:

- **Natural language prefixes:** iom dd ra dhr oxazepam innemen, bij geen resultaat opnieuw contact en consult afspreken
- **Special separator tokens and categorical value tokens:** i.o.m dd ra dhr naar post komen ter controle.
- **Special separator tokens:** i.o.m dd ra dhr naar post komen ter controle.

None of the approaches results in an advice that includes tramadol, which was incorrect. The advice resulting from the approach with natural language prefixes now mentions oxazepam, which makes more sense than tramadol, but was also not mentioned by the domain expert.

For each of the approaches the advice may not have changed, improved or became worse. We consider an advice to not have changed if the advice is identical or is simply rephrased with no new information. An advice is considered to have improved if it for instance includes information that is mentioned in the comments made by the domain expert. An advice is considered worse if it is for example a safety net is not included anymore. Table 6.10 shows an example where natural lanuage prefixes slightly improves the generated advice, since it includes putting a cold cloth on the neck, which was also proposed by the domain expert. The other two approaches lead to a worse advice in this example, because they simply refer to another source for the advice. Table 6.11 shows another example. Here, all approaches lead to an improved advice, since they all recognize that the crisis service/hotline should be called, instead of saying there are no alarm symptoms, which is also mentioned by the domain expert.

TABLE 6.10: Example of outputs using the different approaches from experiment 7 on a case were the domain expert disagreed with original output. Outputs are translated from Dutch to English.

Original Ouput	Blow your nose thoroughly and pinch it shut. if no result then contact own GP
Comment domain expert	Keep head slightly bent forward. Blow nose thoroughly. Hold nose under nasal bone tightly closed for about 10 min. Put a cold wet cloth on the neck. Repeat procedure if bleeding has not stopped. If the bleeding still has not stopped, contact the doctor again and a consultation at the HAP will be scheduled.
NL Prefixes	for now first blow nose well and put cold cloth on neck if nosebleed does not stop then call hap back
Special separator tokens / categorical value tokens	advice given according to thuisarts.nl
Special separator tokens	See thuisarts.nl nosebleed

We also counted how often the advice did not change, improved and got worse for each approach. These counts are given in table 6.12. We observe that in most cases there is no change between the output that was presented during the user study and the advice generated by the 3 new approaches from this experiment. For each approach 3 or 4 advice improved, while 2 or 3 got worse. So, these approaches can lead to both better and worse advice.

TABLE 6.11: Example of outputs using the different approaches from experiment 7 on a case were the domain expert disagreed with original output. Outputs are translated from Dutch to English.

Original Ouput	Advice: no alarm symptoms for now and tomorrow contact own GP. Safety net: If symptoms worsen contact again.
Comment domain expert	Assessment crisis service given obvious suicidal manifestations
NL Prefixes	Advice: Crisis service called they are going to contact her
Special separator tokens / categorical value tokens	Advice: Reporting center called they are going to contact
Special separator tokens	Advice: Called reporting center confused persons they are going to make contact

There are a couple of things to note. For some of the cases where the domain experts disagreed, this was because they felt like they were missing information such as age, to give a proper judgement. Furthermore, there are cases where the domain experts disagreed with the urgency level itself, which is not an output of the model.

TABLE 6.12: Counts of advice that remained unchanged, improved or gotten worsen for different input processing approaches. These counts are for cases where domain experts originally disagreed or strongly disagreed with the generated advice.

	No Change	**Improved**	**Worse**
Natural language prefixes	24	4	3
Special separator tokens / categorical value tokens	26	3	2
Special separator tokens	25	4	2

Given the limited number of changes in advice compared to the advice provided in the user study cases, it is inconclusive whether these approaches lead to an improvement in model performance. Nevertheless, they do show improved performance in terms of ROUGE scores. The differences in ROUGE scores among the approaches from this experiment are minimal. Considering these marginal differences and the fact that using only special token prefixes (excluding categorical value tokens) demands the least resources, which is also shown by number of required epochs, we conclude that this is the best setup for pre-processing the input.

Chapter 7

User study (*SQ4* / *SQ5*)

To assess the usefulness of our model in practice we conduct a user study. This user study has several goals:

1. To assess whether the model produces appropriate advice.
2. To assess whether the model outputs can (partly) be used in practice.
3. To assess the usefulness of our explainability method in practice.

To this end, we have our participants work through several cases. Each case consists of a sample data point from our test set. However, the unstructured text advice has been replaced with the output of our model. For each case the participants answers 2 questions. These questions are shown in figure 7.1. Statement 1 translates to: Given the triage information the given 'own advice' is appropriate. This question is answered using a Likert scale from 1 (Completely Disagree) to 5 (Completely Agree) and aims to achieve goal 1 of the user study. To achieve goal 2 of the user study, question 2 asks the participant to change the given 'own advice' if they deem this necessary. This way we can assess if the generated advice is (partly) usable, which could still reduce the typing workload of triagists.

To achieve goal 3, aggregated input attributions as explained in section 5.6.2, are added to 5 words that have the highest attributions. The number of steps for approximating the integrated gradients were set to 25. An example of what this coloring looks like in one of the cases, is given in figure 7.2.

At the end of the cases, 4 general questions are asked:

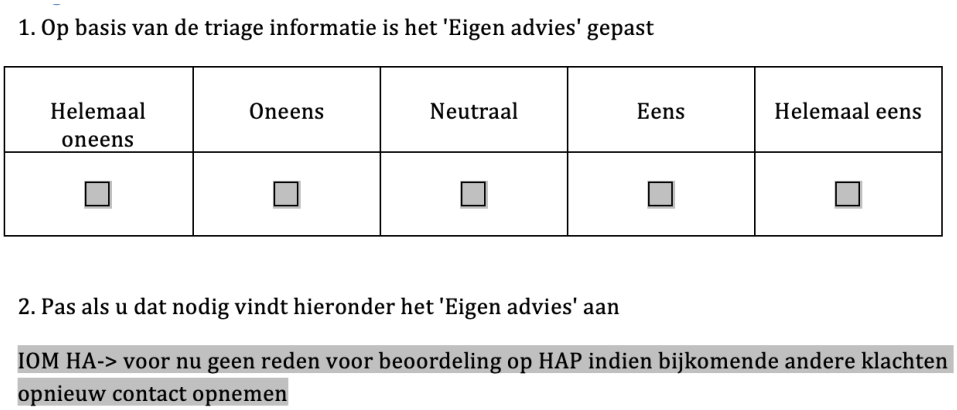


FIGURE 7.1: Questions asked for a case in the user study

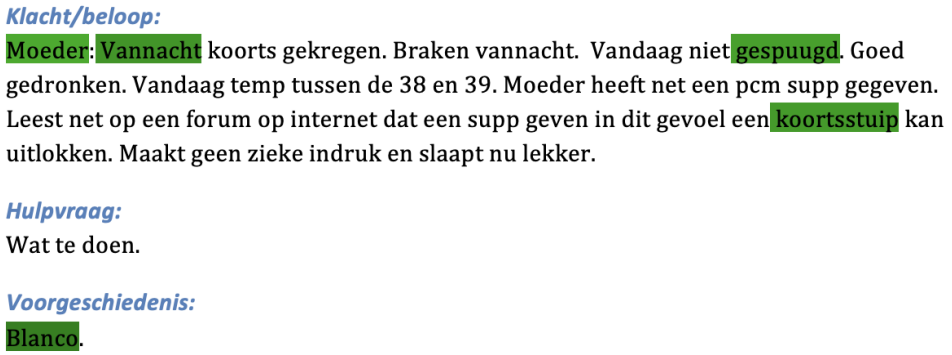


FIGURE 7.2: Example of colored words with high attribution in a case of the user study

1. *Was it difficult to assess whether the advice was appropriate based on the given information? Why or why not?* It could be that the participant does not always have enough information to answer the questions of a case. We aim to capture this with this question.

2. *Could the suggested advice you evaluated (which was provided under 'Own advice') help you formulate 'Own advice' more quickly? Why or why not?* Even if a participants think that the generated advice are appropriate, it could still be that they do not think it will help them.

3. *Did the colored words in the triage information assist you in determining whether you found 'Own advice' suitable? Why or why not?* This question tries to assess the usefulness of our attribution method during decision making.

4. *Did the colored words give you more confidence in the suggested 'Own advice'? Why or why not?* This question aims to assess whether the participant has more trust in the output due to the attribution method.

For this study 4 triagists and 3 regieartsen were recruited. A "regiearts" is a general practitioner (GP), who is the point of contact for triagists at an out-of-hours primary care center in case of questions. The "regiearts" is also ultimately responsible for approving every advice that is given during his shift.

In total 185 unique cases were sampled from our test set. Categorical features were included and the All-Text approach from experiment 1 of section 6.3 was used when preparing these samples. For each sample an output was generated using the Rob2Rob model with weight sharing. Beam search was used with a beam width of 3 and the repetition penalty was set to 1.2.

Each participant was asked to assess a minimum of 15 cases, to limit their workload. We assigned 3 triagists and 3 GPs the same set of 5 cases. This was done to assess the variability between individuals. The other 10 cases of these participants will be unique. The remaining triagist received 15 unique cases. Each partcipant also received 15 additional unique cases which are optional. These are optional to prevent people from refusing to participate because of a high workload.

7.1 Results case questions

In total 160 cases where assessed by domain experts. Of which 5 cases were shared between 6 out of 7 participants. An overview of the number of cases assessed is given in table 7.1.

TABLE 7.1: Number of shared and unique cases assessed by each participant

	Shared cases assessed	**Unique cases assessed**
Triagist 1	0	15
Triagist 2	4	22
Triagist 3	5	10
Triagist 4	5	10
GP 1	5	25
GP 2	5	25
GP 3	5	24
Total	29	131

For the 5 cases that were shared between participants, table 7.2 shows the responses to the statement *"Given the triage information the given advice is appropriate"*. For case 2 and 5 the participants have a similar assesment where the maximum difference is 1 point on the Likert scale. For case 2 almost all participants (strongly) disagree with the generated advice. The generated advice and changes by domain experts for this case are given in table 7.4. We can see that the generated advice is really short and not specific at all. There

TABLE 7.2: Likert scale responses for the 5 cases that were shared across participants. 1 is 'Strongly Disagree' and 5 is 'Strongly Agree'.

	Case 1	**Case 2**	**Case 3**	**Case 4**	**Case 5**
Triagist 2	3	1	3	2	-
Triagist 3	4	2	2	2	4
Triagist 4	2	2	2	4	4
GP 1	5	1	2	3	4
GP 2	5	2	4	2	3
GP 3	4	2	4	3	3

are differences in the changes that different domain experts make. GP 1/2 do not agree with the urgency level and respond that they would let the child come to the center for an assessment. Triagist 2/3 and GP 3 add a safety net to the advice, while triagist 4 changes the advice itself. This shows that domain experts may have different ideas about how an advice should be changed in case they disagree with it.

For the remaining unique cases, so the ones who are not shared between participants, the responses to the statement: *"Given the triage information the given advice is appropriate"*, are given in figure 7.3.

In 47% of the cases the participants agreed with the statement that the advice was appropriate and in 21% they strongly agreed. This indicates that the model is capable of producing appropriate advice. For most of these cases the generated advice is simple and mention that there are no alarming signals and give a safety net in case symptoms worsen. An example of a generated advice, translated from Dutch to English, in this category is: *"for now, no alarm signals; in case of shortness of breath or new symptoms, contact again"*. For these cases where the domain experts agree, the generated advice can

TABLE 7.3: Changes made by domain experts to the advice from case 1. The advice is translated from Dutch and Domain experts that did not make changes are omitted.

Generated advice	IOM HA-> for now no reason for assessment at HAP if additional other complaints, contact again
Triagist 2	Iom RA -> U3-> U5 exp. If persistent headache symptoms contact own gp tomorrow. Safety net: if increase in headache symptoms despite pcm , vomiting or other worsening call back HAP.
Triagist 3	Addition: Indicate if persistent headache to contact own GP. IOM GP-> for now no reason for assessment at HAP if additional other symptoms contact again
Triagist 4	Missing the age of the patient in question. Based on age, an assessment at the HAP may be necessary. This especially for the elderly.
GP 3	IOM GP-> for now no reason for assessment at HAP if additional other symptoms re-contact. I would give a wake-up advice and clearly refer to thuisarts.nl for proper implementation of this

TABLE 7.4: Changes made by domain experts to the advice from case 2. The advice is translated from Dutch.

Original generated advice	For now no action :
Triagist 2	Iom RA: exp. if within 6 hours no good diaper re-contact.
Triagist 3	Safety net according to contact advice fever child.
Triagist 4	Administering small bits of drink/fluids advised For now, no action :
GP 1	Based on longer than 3 days of fever, sick child and moderately wet diaper assessment at HAP and do not wait until after weekend
GP 2	Child makes a sick impression, suspicion of dehydration, in my opinion assessment at HAP, U3
GP 3	No action for now but contact if symptoms increase, less alertness, etc.

also be more complex such as for example: *"iom doctor: now inject 6eh novorapid and in 2 hours measure blood sugar again and call through."* or *"Advice: get prescription from service pharmacy. If symptoms do not disappear after the cure, have urine checked by your doctor."*. So, for most of the cases where the domain experts (strongly) agree, the generated advice is straightforward, but more complex advice is also found.

In 13% of the cases the participants were neutral with regards to the statement, in 18% they disagreed and in 2% of the cases they strongly disagreed. An example of a case where the participant disagreed is given in table 7.5. The generated advice suggests to get microlax, which is a laxative, but the domain expert thinks the patient should get a stronger laxative. Another example where the participant disagreed is given in table 7.6.

Here the advice suggests ibuprofen for the pain, but according to the domain expert, this is not allowed due to pregnancy. So we see that the model can struggle with subtleties such as pregnancy, which leads to an incorrect advice.

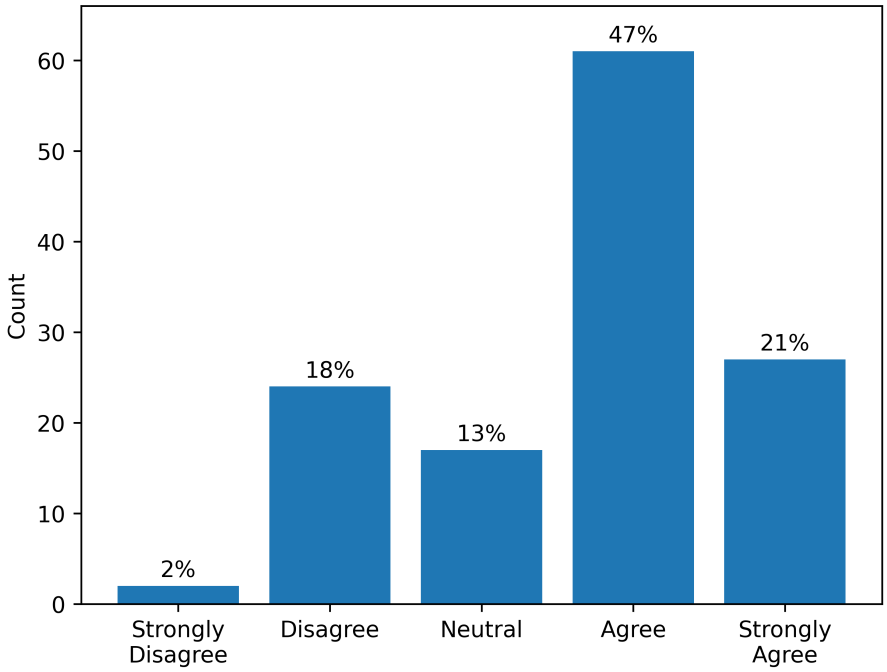


FIGURE 7.3: Counts of Likert scale responses to the statement *"Given the triage information the given advice is appropriate"*, for all the cases that were unique across all participants.

TABLE 7.5: Example of a case where the domain expert disagreed with generated advice. Translated from Dutch.

Symptoms	Was in the hospital last week because of a tumor in the head. Today home again. Can't pass stool. had too little movement. constantly has small bits of stool. May not push too much.
Question	Would like something to promote defecation.
Generated advice	for now get a microlax from the service pharmacy. Call back if no result or symptoms worsen.
Domain expert advice	In my opinion something stronger to laxate, such as phosphate enema

The participants were also asked to change the advice if they deemed it necessary. It is difficult to quantify how much they changed the advice, because the participants used different approaches. For example, some participants kept the output of the model, but added *"Addition: ..."*, indicating what they were missing. While others also wrote inside the advice why they changed the advice. Therefore, we attempted to manually find patterns in the changes that were made. In general, we observed 5 reasons that the domain experts had for making changes to the output of the model. These reasons and how often they occurred are given in table 7.7. Note that more than one point can occur in a change. For

TABLE 7.6: Example of a case where the domain expert disagreed with generated advice. Translated from Dutch.

Symptoms	is 38wk pregnant. has hemorrhoids own gp made incision, had thrombosis hemorrhoid, pain heavier and is still there. difficulty sitting. This morning by own gp still incised, pcm does not help sufficiently now. struggles to go to toilet
Question	what can we do? don't want to go into night like this
Generated advice	iom ra > for now no alarm signals, stand by with painkillers pcm and ibuprofen. if acutely sicker or other pain then call hap. otherwise own gp tomorrow for further management. mw reassured and agreed
Domain expert comment	Patient not allowed to take ibuprofen due to pregnancy

example a domain expert can make both an addition to the advice and also add a safety net.

TABLE 7.7: Changes made by domain experts to the generated outputs of the cases that were unique across participants

#	**Reason for changing the output**	**Count**
1	A safety net was missing	23
2	An addition to the advice was needed	6
3	They disagreed with the advice	22
4	They were missing necessary information	3
5	They disagreed with the urgency	9

Outputs of the model that lead to reason 1 (which occurred most often) and reason 2 could still be valuable, because the participants kept the output of the model, but simply added something extra. Therefore, they still have to type less when they use the output of the model. Model outputs that resulted in point 3, which occur second most often, are not useful, since the triagist would be spending time changing the advice, which probably takes more time than writing the advice from scratch. Point 4 is a shortcoming of the user study itself, since some information was lacking such as age. In practice, a triagist would, almost always, request such information from a different system using the citizen service number of the person that is calling. Point 5 shows that domain experts may not agree with urgency level of our input data, which is something out of our control. This also shows that certain cases are difficult to assess for humans as well.

7.2 Results general questions

The general question were answered by 2 triagists and 3 general practitioners. All the general questions were open-ended, but they did steer towards a yes or no answer with explanation. Therefore, we tried to deduce a yes/no answer for each question. These answers are given in table 7.8. The raw answers to the each question can be found in

appendix B.

TABLE 7.8: Answers to the general questions

	Question 1	**Question 2**	**Question 3**	**Question 4**
Triagist 1	Yes	Sometimes	No	No
Triagist 2	Yes	No	No	No
GP 1	No	No	No	No
GP 2	Yes	Yes	Yes	Yes
GP 3	Yes	No	No	No

Triagist 1/2 and GP 1/3 sometimes found it challenging to assess the advice, because they were missing some information such as age. GP 2 found it hard to assess the cases, because each healthcare professional has his/her own formulation.

With regard to the usefulness of the advice that were given by the model, triagist 2 mentioned that the advice were sometimes incomplete and really often lacked a concrete safety net. For example, for the generated advice: *"Iom RA: no alarm symptoms now, Monday EHA."*, this triagist is missing the safety net: *"In case increase in abdominal pain, vomiting call back HAP"*. Triagist 1 did not think the model outputs would help her because of her way of working, but she could imagine that it could be useful for others. GP 1 mentioned his/her concerns about confirmation bias. GP 2 responded that the model outputs could save time when composing an advice and that small changes can easily be made. GP 3 said it could also be confusing and in his/her opinion relying on your own judgment is better and more objective.

As for the colored words of our attribution method, GP 2 found it useful because he could focus on the important parts while ignoring noise. GP 1 did not find it useful, because it sometimes puts emphasis on irrelevant things. He mentioned that for the case with the following symptoms: *went through foot this morning; walked out, sagged through foot, slip? is swelling, not cooled, sat on couch, limping to toilet; toes a little tingly*, the attribution method had for example marked *"amputution injury"* inside the questions and answers section of the triage information, which is obviously irrelevant for these minor symptoms. GP 3 found the colored words annoying. Triagist 2 disregarded the colored words completely, while triagist 1 found it distracting, because he/she immediately focused on those words and felt like she did not read the entire case properly. This suggests that the method could have been more useful if the attention was directed towards the right words instead of irrelevant ones.

7.3 Conclusion

A user study, involving 4 triagists and 3 GPs, was conducted to assess the appropriateness and usefulness of the advice produced by our model. Furthermore, the study aimed to assess the effectiveness of the proposed explainability method in practice.

The results of the study showed that in the majority of cases the model produced an appropriate advice. However, in some cases the advice was inappropriate and also often lacked information such as a safety net. Even though not all participants answered the general questions, the study also revealed that the majority of participants indicated that, in general, they did not think that the outputs would be useful in practice. These conflicting results from the case questions and general questions show that there is still a lot of room for improvement. Finally, the study also showed that the explainability method in its current form was not beneficial for most participants.

In general, we can conclude that the model is capable of producing appropriate advice, but domain experts have doubts about its usefulness in practice. Furthermore, the explainability method is not effective in its current form.

Chapter 8

Conclusions

VIPLive Spoed EPD is a software application developed by Topicus that is used by triagists to enter information about a triage conducted by phone at an out-of-hours primary care center. Due to a constant shortage of staff, a triage has to be conducted quickly. In some cases an advice is given in the form of unstructured text. The goal of this research was to reduce the workload of triagists by predicting this natural language advice based on the input given by the person who calls the care center. Here, we answer each sub-research question and the main research question.

- ***SQ1****: What is the performance of a domain-specific PLM against a domain-generic PLM?*

  We proposed the healthcare-specific Med2Med and domain-generic Rob2Rob PLM, with identical setups apart from the model checkpoints. Rob2Rob slightly outperformed Med2Med on our dataset, showing that healthcare-specific pre-training on hospital notes is not beneficial compared to domain-generic pre-training in a triage setting at out-of-hours primary care centers. There are possible explanations for this. Firstly, the size of the pre-training dataset for Med2Med is significantly smaller compared to that of Rob2Rob. Furthermore, the natural language in our dataset likely contains fewer medical terms in comparison to the hospital notes in which Med2Med was pre-trained.

- ***SQ2****: How can natural language and categorical features be combined to train a PLM that generates triage advice?*

  We trained our proposed Rob2Rob only on features that were originally unstructured text. We observed that this model significantly under performed compared to a Rob2Rob model that was trained on all features using an All-Text approach. Showing that there is valuable information in the categorical features, when these features are simply concatenated as text.

  Next, we proposed a Fuse-Early approach in which separate embeddings are created for categorical features, before fusing them with the unstructured text embeddings. We observed that the performance of Rob2Rob decreases significantly when using the Fuse-Early approach. The model is even outperformed by the Rob2Rob model that only considers natural language text features. This shows that, at least with our dataset, processing categorical features separately is not beneficial. There could be several explanations for this result. First, the size of our dataset might be a limiting factor, given the greater complexity and increased number of parameters that need to be learned from scratch in the Fuse-Early approach. Furthermore, it is possible that

our Fuse-Early architecture is suboptimal because our input does not pass through the pre-trained embedding layer of Rob2Rob, and we have to project our categorical embeddings to the hidden dimension of Rob2Rob. Previous work [47] did not find such a big difference in performance between All-Text and Fuse-Early, which indicates that a deficiency in our architecture is more plausible.

In an effort to enhance the context provided by the All-Text approach, we proposed three strategies. Firstly, we included natural language prefixes that describe each feature in front of the corresponding input feature. Secondly, we introduced a special token for each input feature into the tokenizer and placed it in front of the respective feature. Finally, we extended the last approach by introducing a token for each possible value of the categorical features. Each approach improved performance in terms of ROUGE scores, and the differences between these approaches were marginal. Considering these minor differences, along with the fact that using only special token prefixes leads to the smallest memory footprint, we conclude that this is the best setup for preprocessing the input among all the strategies we explored for combining input features.

- ***SQ3****: What strategies with regard to the setup of the PLM can be used to improve its performance to generate advice?*

  First of all, to reduce the memory footprint of our model and to decrease training time, we used weight sharing between the encoder and decoder of our Rob2Rob model. This only led to a small decrease in performance in terms of ROUGE score, which is in line with previous work [45]. Secondly, introducing a repetition penalty successfully mitigated the problem of consecutive uni- and bi-gram repetition. Finally, using beam search with a beam width of 3 improved the performance of our Rob2Rob model when using an All-Text approach.

- ***SQ4****: How can we evaluate the generated advice?*

  We used ROUGE, which measure the syntactic similarity between two pieces of text, to compare the performance of different models and different model setups. However, to evaluate the usefulness of the model outputs we recruited domain experts. By working through several cases we had them assess the appropriateness of the model outputs. We observed that they primarily agreed with the advice that was given by the model. Furthermore, through general open-ended questions we also found that they have doubts about the usefulness of the outputs in practice. However, the number of participants and number of cases assessed in our user study was low. Therefore, these findings are not sufficient to statistically evaluate the usefullness of our model in practice.

- ***SQ5****: How can we add explainabilty to the generated advice?*

  We proposed an attribution method that aggregates input attributions that are obtained using integrated gradients. This method was applied on the cases of the user study where words with high attributions were marked. However, the majority of domain experts did not find this method useful because it was found to be distracting, as it occasionally marked irrelevant words. Therefore, we conclude that the proposed attribution method does not provide a good explanation of the model output to the end-user.

So, to answer our main ***RQ***, we can conclude that our proposed and finetuned Rob2Rob PLM, in most cases, is capable of generating appropriate advice in a triage setting at

out-of-hours primary care centers. However, domain experts have doubts about their usefulness in practice. Furthermore, our proposed attribution method was not useful as an explainabilty method according to domain experts.

Here, we also discuss some limitations and future work. First of all, our models were trained on data that was produced by a single out-of-hours primary care center. Although all centers are expected to adhere to the same standard (NTS), it could be that there might be subtle differences in the way of working or writing styles between different centers. Therefore, we cannot guarantee that our trained models generalize well for other centers. As such, it would be beneficial to train the models on more data from different out-of-hours primary care centers.

Secondly, the proposed attribution method has its limitations. Integrated gradients results in scores for each input token across all output tokens. We aggregated these scores by summing the attribution vectors for each input token. However, since we use a conditional language model, this may not be fair because the output tokens depend on each other. Furthermore, we summed the aggregated attribution scores of the input tokens that together form a word to make it more readable for the end user. Therefore, words that are made up of multiple tokens are more likely to have a higher attribution score, which may not be desirable. So, in hindsight, it may have been better to average the attribution scores over the tokens that form a word. In general, a big limiting factor for explaining our model outputs is the fact that we have multiple outputs (a sequence of tokens). This makes it hard to have an easily digestible explanation, because it is hard to explain why the model produces the entire output as a whole.

Furthermore, we restricted ourselves to data that is entered into the Spoed EPD application. However, triagists can retrieve other personal information, such as age, from different systems. Therefore, it could be beneficial to also add this information to the input of the model, such that is has more context.

Also, the conducted user study has its limitations. The participants were only presented with the information that they themselves enter into the system. In practice they also request personal information, such as age, from a different system. This sometimes made it difficult for them to assess the advice. Furthermore, 136 different samples from our test set were evaluated by domain experts. This is obviously a small portion of the entire test set, which consisted of 12076 samples. Therefore, to really assess the usefulness of the model in practice, it would be valuable to conduct a pilot study where the model is actually integrated into the application.

Finally, our user study showed that domain experts have doubts about the usefulness of the model outputs in practice, while they did assess that they were appropriate in most cases. Therefore, instead of presenting the advice as a suggestion, the model can also be used to create an auto-complete system, as also suggested by Liu et al. [37], because it is trained to predict the next token given some context.

Bibliography

[1] Nederlandse triage standaard (nts). https://de-nts.nl/. Accessed: 14-03-2023.

[2] J Alammar. Ecco: An open source library for the explainability of transformer language models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*. Association for Computational Linguistics, 2021.

[3] Émilien Arnaud, Mahmoud Elbattah, Maxime Gignon, and Gilles Dequen. Nlp-based prediction of medical specialties at hospital admission using triage notes. In *2021 IEEE 9th International Conference on Healthcare Informatics (ICHI)*, pages 548–553, 2021.

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[5] Satanjeev Banerjee and Alon Lavie. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan, June 2005. Association for Computational Linguistics.

[6] Zina Ben Miled, Paul R. Dexter, Randall W. Grout, and Malaz Boustani. Feature engineering from medical notes: A case study of dementia detection. *Heliyon*, 9(3):e14636, 2023.

[7] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166, 1994.

[8] Alexander Binder, Grégoire Montavon, Sebastian Lapuschkin, Klaus-Robert Müller, and Wojciech Samek. Layer-wise relevance propagation for neural networks with local renormalization layers. In *Artificial Neural Networks and Machine Learning – ICANN 2016*, pages 63–71, Cham, 2016. Springer International Publishing.

[9] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D3: Data-driven documents. *IEEE Trans. Visualization & Comp. Graphics (Proc. InfoVis)*, 2011.

[10] Min-Chen Chen, Ting-Yun Huang, Tzu-Ying Chen, Panchanit Boonyarat, and Yung-Chun Chang. Clinical narrative-aware deep neural network for emergency department critical outcome prediction. *Journal of Biomedical Informatics*, 138:104284, 2023.

[11] Kyunghyun Cho, Bart van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar, October 2014. Association for Computational Linguistics.

[12] Eldan Cohen and J. Christopher Beck. Empirical analysis of beam search performance degradation in neural sequence models. In *International Conference on Machine Learning*, 2019.

[13] Alister D'Costa, Stefan Denkovski, Michal Malyska, Sae Young Moon, Brandon Rufino, Zhen Yang, Taylor Killian, and Marzyeh Ghassemi. Multiple sclerosis severity classification from clinical text. In *Proceedings of the 3rd Clinical Natural Language Processing Workshop*, pages 7–23. Association for Computational Linguistics, November 2020.

[14] Wietse de Vries, Andreas van Cranenburgh, Arianna Bisazza, Tommaso Caselli, Gertjan van Noord, and Malvina Nissim. Bertje: A dutch bert model, 2019. arXiv:1912.09582.

[15] Pieter Delobelle, Thomas Winters, and Bettina Berendt. RobBERT: a Dutch RoBERTa-based Language Model. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 3255–3265. Association for Computational Linguistics, November 2020.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

[17] Auke Elfrink, Iacopo Vagliano, Ameen Abu-Hanna, and Iacer Calixto. Soft-prompt tuning to predict lung cancer using primary care free-text dutch medical notes. In *Artificial Intelligence in Medicine*, pages 193–198, Cham, 2023. Springer Nature Switzerland.

[18] Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 889–898, Melbourne, Australia, July 2018. Association for Computational Linguistics.

[19] Yanjun Gao, Dmitriy Dligach, Timothy Miller, Dongfang Xu, Matthew M. M. Churpek, and Majid Afshar. Summarizing patients' problems from hospital progress notes using pre-trained sequence-to-sequence models. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 2979–2991, Gyeongju, Republic of Korea, 2022. International Committee on Computational Linguistics.

[20] Divya Gopinath, Monica Agrawal, Luke Murray, Steven Horng, David Karger, and David Sontag. Fast, structured clinical documentation via contextual autocomplete. In *Machine Learning for Healthcare Conference*, pages 842–870. PMLR, 2020.

[21] Marjolein Groenendijk. Drukte op huisartsenpost door onnodige belletjes: 'kind met keelpijn is geen spoed'. *Het AD*. URL: https://www.ad.nl/dordrecht/drukte-op-huisartsenpost-door-onnodige-belletjes-kind-met-keelpijn-is-geen-spoed~a0d0d113/.

[22] Cheng Guo and Felix Berkhahn. Entity embeddings of categorical variables, 2016. arXiv:1604.06737.

[23] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[24] Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020.

[25] Yun Hu, Guokai Han, Xintang Liu, Hui Li, Libao Xing, Yong Gu, Zuojian Zhou, and Haining Li. Design and implementation of a medical question and answer system based on deep learning. *Mathematical Problems in Engineering*, 2022:1–6, 09 2022.

[26] Kexin Huang, Jaan Altosaar, and Rajesh Ranganath. Clinicalbert: Modeling clinical notes and predicting hospital readmission, 2020. arXiv:1904.05342.

[27] Xin Huang, Ashish Khetan, Milan Cvitkovic, and Zohar Karnin. Tabtransformer: Tabular data modeling using contextual embeddings, 2020. arXiv:2012.06678.

[28] Hadi Jahanshahi, Syed Kazmi, and Mucahit Cevik. Auto response generation in online medical chat services. *Journal of Healthcare Informatics Research*, 6(3):344 – 374, 2022.

[29] Baoyu Jing, Pengtao Xie, and Eric Xing. On the automatic generation of medical imaging reports. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2018.

[30] Neel Kanwal and Giuseppe Rizzo. Attention-based clinical note summarization. In *Proceedings of the 37th ACM/SIGAPP Symposium on Applied Computing*, SAC '22, page 813–820, New York, NY, USA, 2022. Association for Computing Machinery.

[31] Nitish Shirish Keskar, Bryan McCann, Lav R. Varshney, Caiming Xiong, and Richard Socher. Ctrl: A conditional transformer language model for controllable generation, 2019. arXiv:1909.05858.

[32] Yoojoong Kim, Jong-Ho Kim, Young-Min Kim, Sanghoun Song, and Hyung Joo. Predicting medical specialty from text based on a domain-specific pre-trained bert. *International Journal of Medical Informatics*, 170:104956, 12 2022.

[33] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[34] Jinhyuk Lee, Wonjin Yoon, Sungdong Kim, Donghyeon Kim, Sunkyu Kim, Chan Ho So, and Jaewoo Kang. BioBERT: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240, sep 2019.

[35] Yikuan Li, Shishir Rao, José Roberto Ayala Solares, Abdelaali Hassaine, Rema Ramakrishnan, Dexter Canoy, Yajie Zhu, Kazem Rahimi, and Gholamreza Salimi-Khorshidi. Behrt: transformer for electronic health records. *Scientific reports*, 10(1):1–12, 2020.

[36] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain, July 2004. Association for Computational Linguistics.

[37] Peter J. Liu. Learning to write notes in electronic health records, 2018. arXiv:1808.02622.

[38] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach, 2019. arXiv:1907.11692.

[39] Andriy Mulyar, Ozlem Uzuner, and Bridget McInnes. MT-clinical BERT: scaling clinical information extraction with multitask learning. *Journal of the American Medical Informatics Association*, 28(10):2108–2115, 08 2021.

[40] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA, July 2002. Association for Computational Linguistics.

[41] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2019. Curran Associates Inc.

[42] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21:140:1–140:67, 2020.

[43] Afsana Binte Rakib, Esika Arifin Rumky, Ananna J. Ashraf, Md. Monsur Hillas, and Muhammad Arifur Rahman. Mental healthcare chatbot using sequence-to-sequence learning and bilstm. In *Brain Informatics*, pages 378–387, Cham, 2021. Springer International Publishing.

[44] Max Tigo Rietberg, Van Bach Nguyen, Jeroen Geerdink, Onno Vijlbrief, and Christin Seifert. Accurate and reliable classification of unstructured reports on their diagnostic goal using bert models. *Diagnostics*, 13(7):1251, 2023.

[45] Sascha Rothe, Shashi Narayan, and Aliaksei Severyn. Leveraging pre-trained checkpoints for sequence generation tasks. *Transactions of the Association for Computational Linguistics*, 8:264–280, dec 2020.

[46] M. Schuster and K.K. Paliwal. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

[47] Xingjian Shi, Jonas Mueller, Nick Erickson, Mu Li, and Alexander J. Smola. Benchmarking multimodal automl for tabular data with text fields. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks*, 2021.

[48] Avanti Shrikumar, Peyton Greenside, and Anshul Kundaje. Learning important features through propagating activation differences. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 3145–3153. JMLR.org, 2017.

[49] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep inside convolutional networks: Visualising image classification models and saliency maps. In *2nd International Conference on Learning Representations, Workshop Track Proceedings*, 2014.

[50] Nicholas Sterling, Rachel Patzer, Mengyu Di, and Justin Schrager. Prediction of emergency department patient disposition based on natural language processing of triage notes. *International Journal of Medical Informatics*, 129, 06 2019.

[51] Mukund Sundararajan, Ankur Taly, and Qiqi Yan. Axiomatic attribution for deep networks. In *Proceedings of the 34th International Conference on Machine Learning - Volume 70*, ICML'17, page 3319–3328. JMLR.org, 2017.

[52] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, page 3104–3112, Cambridge, MA, USA, 2014. MIT Press.

[53] Deeksha Varshney, Aizan Zafar, Niranshu Kumar Behera, and Asif Ekbal. Knowledge grounded medical dialogue generation using augmented graphs. *Scientific Reports*, 13(1):3310, 2023.

[54] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA, 2017. Curran Associates Inc.

[55] Stella Verkijk and Piek Vossen. Medroberta.nl: A language model for dutch electronic health records. *Computational Linguistics in the Netherlands Journal*, 11:141–159, Dec. 2021.

[56] Hessel von Piekartz. Huisartsen hebben al te weinig tijd, komen er ook nog mensen met pijnlijke tenen naar de spoedpost. *De Volkskrant*. URL: https://www.volkskrant.nl/nieuws-achtergrond/huisartsen-hebben-al-te-weinig-tijd-komen-er-ook-nog-mensen-met-pijnlijke-tenen-naar-de-spoedpost~bfddc09f/.

[57] Haifeng Wang, Jiwei Li, Hua Wu, Eduard Hovy, and Yu Sun. Pre-trained language models and their applications. *Engineering*, 25:51–65, 2023.

[58] Sean Welleck, Ilia Kulikov, Stephen Roller, Emily Dinan, Kyunghyun Cho, and Jason Weston. Neural text generation with unlikelihood training. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

[59] Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Computation*, 1(2):270–280, 1989.

[60] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45. Association for Computational Linguistics, October 2020.

[61] Weizhe Yuan, Graham Neubig, and Pengfei Liu. Bartscore: Evaluating generated text as text generation, 2021. arXiv:2106.11520.

[62] Hugh Zhang, Daniel Duckworth, Daphne Ippolito, and Arvind Neelakantan. Trading off diversity and quality in natural language generation. In *Proceedings of the Workshop on Human Evaluation of NLP Systems (HumEval)*, pages 25–33. Association for Computational Linguistics, April 2021.

[63] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q. Weinberger, and Yoav Artzi. Bertscore: Evaluating text generation with BERT. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*, 2020.

[64] Xingxing Zhang, Yiran Liu, Xun Wang, Pengcheng He, Yang Yu, Si-Qing Chen, Wayne Xiong, and Furu Wei. Momentum calibration for text generation, 2022. arXiv:2212.04257.

[65] Xingyu Zhang, Joyce J. Kim, Rachel Elizabeth Patzer, Stephen R. Pitts, Aaron Patzer, and Justin D Schrager. Prediction of emergency department hospital admission based on natural language processing and neural networks. *Methods of Information in Medicine*, 56:377 – 389, 2017.

Appendix A

Evaluation metrics

Model	Fusion Strategy	Only free-text features	encoder decoder weights shared	Epochs	R1 F1	R2 F1	BLEU	METEOR
Rob2Rob	All-Text	No	Yes	3	27.5	12.0	7.5	24.7
Med2Med	All-Text	No	Yes	3	27.2	11.8	7.3	24.5
Rob2Rob	All-Text	Yes	Yes	4	22.0	8.1	4.7	19.9
Rob2Rob	Fuse-Early	No	Yes	5	18.4	3.6	1.3	18.3
Rob2Rob	All-Text	No	No	3	27.9	12.4	7.7	25.0

TABLE A.1: All evaluation metrics on the test set of the models trained in experiments 1 - 4 of chapter 6. R1 is ROUGE-1, R2 is ROUGE-2.

Appendix B

Answers to general questions of user study

1. Was het lastig om te beoordelen of het advies passend was op basis van de gegeven informatie? Waarom wel/niet? (Was it difficult to assess whether the advice was appropriate based on the given information? Why or why not?)

    - Triagist 1: Ja vond ik wel, we zijn geneigd om veel te vragen en te willen weten. Dus ik had het gevoel advies te moeten geven op een half verhaal. Dat is natuurlijk lang niet altijd zo. Is meer de aart van het beestje , ervaring en onderbuik gevoel.
    - Triagist 2: Ja, ik mist soms nog info.
    - GP 1: Over het algemeen niet moeilijk om een oordeel te geven, maar hier en daar miste ik wel wat info, bv over de leeftijd van patiënten
    - GP 2: Redelijk lastig. Als zorgverlener kies je je eigen formulering.
    - GP 3: Ja, want leeftijden stonden er veelal niet bij. Soms ook wat tegenstrijdig verhaal

2. Zouden de voorgestelde adviezen die u heeft beoordeeld (en welke gegeven waren onder 'Eigen advies'), u kunnen helpen om sneller een 'Eigen advies' op te stellen? Waarom wel/niet? (Could the suggested advice you evaluated (which was provided under 'Own advice') help you formulate 'Own advice' more quickly? Why or why not?)

    - Triagist 1: Mij niet echt denk ik. Ik klik nu al vaak het hokje bij zelfzorg aan en dan een summier extra regel erbij in mijn eigen bewoording. Denk dat het anderen zeker kan helpen, de een is nu een maal sneller met afhandelen van een melding dan de ander.
    - Triagist 2: De opgestelde adviezen waren soms niet compleet en miste heel vaak een concreet vangnet.
    - GP 1: Helpen niet echt, en ik vraag me af of er door het gegeven advies niet een risico van conformation bias insluipt
    - GP 2: Jazeker. Kan tijd schelen en enige aanpassing tekst is snel verricht.
    - GP 3: Kan ook verwarrend werken; afgaan op eigen oordeel wat mij betreft beter en meer objectief

3. Hielpen de gekleurde woorden in de triage informatie u bij het bepalen of u een 'Eigen advies' passend vond? Waarom wel/niet? (Did the colored words in the triage information assist you in determining whether you found 'Own advice' suitable? Why or why not?)

    - Triagist 1: Nee, het gaf mij afleiding. Ik keek meteen naar de gekleurde woorden en had de neiging om minder goed te lzen daardoor.
    - Triagist 2: Niet opgelet.
    - GP 1: Niet. Soms brenhen ze een probleem terecht onder de aandacht (bv nierdonatie in casus 29), maar soms lijken ze ook niet relevant (amputatieletsel in casus 30)
    - GP 2: Jazeker. Je kunt snel inzoomen op de kern.
    - GP 3: nee, eerder irritant

4. Had u door de gekleurde woorden meer vertrouwen in het voorgestelde 'Eigen advies'? Waarom wel/niet? (Did the colored words give you more confidence in the suggested 'Own advice'? Why or why not?)

    - Triagist 1: Nee eigenlijk een overlap met de vorige vraag. Daarnaast is het denk ik aan de andere kant ook een kwestie van wennen. Bij iedere release moet je ook wennen aan nieuwe dingen. Daarnaast is het een kwestie van vertrouwen hebben. Als je het aangegeven zelfzorg advies mondeling toegelicht hebt hoef je het in principe hierdoor niet uit te schrijven.
    - Triagist 2: Niet opgelet.
    - GP 1: Zie bovenstaande antwoord: nee
    - GP 2: Ik denk het wel. Ontdaan van allerlei ruis in de tekst.
    - GP 3: nee, niet door laten leiden