**CAN THO UNIVERSITY**

**COLLEGE OF INFORMATION AND COMMUNICATION TECHNOLOGY**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**🙞 🕮 🙜**

**Logo

Description automatically generated**

**GRADUATION THESIS**

**BACHELOR OF ENGINEERING IN**

**INFORMATION TECHNOLOGY**

**(HIGH-QUALITY PROGRAM)**

**MEDBOT: CHATBOT ABOUT**

**HEALTHCARE**

**Student: Nguyen Trung Tam**

**Student ID: B1910697**

**Class: 2019-2023 (K45)**

**Advisor: Dr. Lam Nhut Khang**

**Can Tho, 12/2023**

**CAN THO UNIVERSITY**

**COLLEGE OF INFORMATION AND COMMUNICATION TECHNOLOGY**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**🙞 🕮 🙜**

**Logo

Description automatically generated**

**GRADUATION THESIS**

**BACHELOR OF ENGINEERING IN**

**INFORMATION TECHNOLOGY**

**(HIGH-QUALITY PROGRAM)**

**MEDBOT: CHATBOT ABOUT**

**HEALTHCARE**

**Student: Nguyen Trung Tam**

**Student ID: B1910697**

**Class: 2019-2023 (K45)**

**Advisor: Dr. Lam Nhut Khang**

**Can Tho, 12/2023**

EVALUATION OF ADVISOR

....................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................................

Advisor

Lam Nhut Khang

ACKNOWLEDGMENTS

I wish to express my deep gratitude and sincere thanks to my professor Lam Nhut Khang– A lecturer at the College of Information and Communication Technology who gave me the golden opportunity to do this wonderful thesis on the topic “MedBot: Chatbot about healthcare”, which also helped me in doing a lot of research and I came to know about so many new things I am really thankful to them. Then I would like to thank the lecturers of Can Tho University, specifically, the lecturers of the College of Information and Communication Technology who taught me invaluable knowledge when I studied.

I am extremely grateful to my family for their love, prayers, and care for the completion of this thesis. I am very much thankful to my friends for their support when I was doing research at Can Tho University.

Sincerely,

Can Tho, 01/12/2023

Nguyen Trung Tam

TABLE OF CONTENTS

[CHAPTER 1: INTRODUCTION 1](#_Toc144840280)

[1.1. Problem 1](#_Toc144840281)

[1.2. History of problem-solving 2](#_Toc144840282)

[1.3. Purpose 3](#_Toc144840283)

[1.4. The objectives and scope 3](#_Toc144840284)

[1.5. Research Methods: 3](#_Toc144840285)

[1.6. Result 4](#_Toc144840286)

[CHAPTER 2: PROBLEM DESCRIPTION 5](#_Toc144840287)

[2.1. Detailed description of the problem 5](#_Toc144840288)

[2.2. Background 5](#_Toc144840289)

[2.2.1. Python 5](#_Toc144840290)

[2.2.2. Tensorflow 7](#_Toc144840291)

[2.2.3. Deep learning 8](#_Toc144840292)

[2.2.4. BART model 8](#_Toc144840293)

[2.2.5. Dataset 12](#_Toc144840294)

[CHAPTER 3: DESIGN AND IMPLEMENT OF THE CHATBOT 14](#_Toc144840295)

[3.1. Overview 14](#_Toc144840296)

[CHAPTER 4: CONCLUSION AND FUTURE WORK 15](#_Toc144840297)

[4.1. Conclusion 15](#_Toc144840298)

[4.2. Future work 15](#_Toc144840299)

[REFERENCES 16](#_Toc144840300)

[APPENDICES 17](#_Toc144840301)

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

**ABSTRACT**

In recent years, the integration of artificial intelligence (AI) and natural language processing (NLP) technologies has ushered in a new era in healthcare with the emergence of medical chatbots. These intelligent conversational agents hold immense promise in transforming the way healthcare services are delivered, improving patient experiences, and optimizing healthcare workflows.

This abstract provides an overview of the evolving landscape of medical chatbots, their key functionalities, and their impact on the healthcare ecosystem. Medical chatbots are designed to engage in dynamic conversations with users, offering personalized health information, symptom assessment, medication reminders, and appointment scheduling. They empower patients with on-demand access to medical guidance, reducing the burden on healthcare professionals and enhancing overall patient engagement.

Furthermore, medical chatbots demonstrate significant potential in improving healthcare outcomes through early symptom detection and continuous monitoring. By analyzing user-provided information and historical health data, these chatbots can identify potential health risks and provide timely recommendations, ultimately contributing to preventive care.

However, the widespread adoption of medical chatbots does not come without challenges. Ensuring data security, privacy compliance, and maintaining a high standard of accuracy and reliability in medical advice are paramount concerns. The need for seamless integration with electronic health records (EHRs) and healthcare information systems is another technical hurdle that must be addressed.

In conclusion, medical chatbots represent a transformative technology in the healthcare domain, offering a myriad of benefits such as improved accessibility, enhanced patient engagement, and streamlined administrative processes. While challenges persist, their potential to revolutionize healthcare delivery and empower individuals to take control of their health cannot be overlooked. As research and development in AI and NLP continue to advance, the future of medical chatbots holds great promise in reshaping the landscape of healthcare for the better.

**OVERVIEW**

Medical Chatbot – a chatbot application for answering questions about medical health using Python and ML

Nowadays, the necessity for health care is becoming more and more important, especially in a technology world, because of the rapid development of IT and the need for fast and accurate information about healthcare. However, there are a lot of questions and problems that traditional healthcare is limited in providing fast and accurate information. The old manual approach needs humans to search for the info and then prepare the answer for the customers.

Medical chatbot was created to provide a useful tool to assist those who want to find and answer questions about medical health care in the most effective, fast, and convenient way. In addition, the chatbot also supports collecting and updating practical healthcare information from doctors to update the chatbot accordingly. Besides, the chatbot also allows users to search for medical information according to their desire to help them find answers and advice quickly.

# INTRODUCTION

## Problem

Nowadays, healthcare is becoming more and more important, especially in a fast-paced technology world where everyone is so busy with their phone and digital devices. Therefore, the necessity to have a medical chatbot to help people have fast and reliable information about healthcare is considerably important. However, the development of medical chatbots has faced several challenges and problems in the healthcare industry, which modern chatbots aim to address. Here are some key problems that have spurred the development of medical chatbots:

1. Healthcare Accessibility: Many individuals face challenges in accessing healthcare services, particularly in remote or underserved areas. Medical chatbots can provide instant, round-the-clock access to basic medical information and advice, bridging the gap in healthcare accessibility.

2. Appointment Scheduling: Booking appointments with healthcare providers can be cumbersome and time-consuming. Chatbots can streamline the appointment scheduling process, making it more convenient for patients and reducing the administrative burden on healthcare facilities.

3. Information Overload: The internet is filled with vast amounts of health information, making it difficult for individuals to find reliable sources. Medical chatbots offer a trusted source of medical information, helping users sift through the noise to find accurate answers to their health-related questions.

4. Health Awareness and Education: Many people lack access to comprehensive health education. Medical chatbots can serve as educational tools, providing users with valuable information about various health topics and promoting health awareness.

While medical chatbots offer innovative solutions to these problems, it's important to recognize that they are not a replacement for professional medical care. They should complement, not substitute, the expertise of healthcare providers. Additionally, ensuring the accuracy of medical information, maintaining data privacy, and addressing ethical concerns remain ongoing challenges in the development and deployment of medical chatbots.

## History of problem-solving

Related chatbots:

Woebot: The conversational agent was built using Decision Tree and appropriate NLP algorithms and needs to be installed as software in a stand-alone computer. All conversational responses are stored in a database. The specific area of application is Cognitive Behavioural Therapy (CBT) for anxiety and mood disorders targeting young adults who are college students. The data for the training chatbot are collected using surveys. The data processing and analysis are carried out by qualitative analysis of the responses from the surveyors. There were several categories on which Woebot could provide therapy. Results showed that there was a significant improvement in the mean PHQ-9 score compared to controls in the mental condition of participants with 2 weeks of therapy

iHelpr: iHelpr is a text-based interactive Chatbot intended to provide mental health support in the workplace. The iHelpr is a web-based self-assessment tool and is reported to be available for six well-being indicators viz. stress, anxiety, depression, sleep, and self-esteem**.** The Chatbot is developed using the bot development framework by Microsoft’s Cognitive Services, “an Application Programming Interface (API) that can process natural language, enable a Chatbot to recognize speech, and image-processing technology”

Tess: Tess is a web-based chatbot devised by X2AI Inc. with an access interface via SMS (on mobile) and Facebook Messenger application.The chatbot is developed based on machine learning algorithms integrated with psycho-educational concepts and is said to be developed in conjunction and collaboration with trained mental health professionals.

SISU: an I ntervention for Self-help to U plift psychological wellbeing chatbot, providing support for therapeutic writing.

## Purpose

The application "MedicalBot: Chatbot about healthcare" is used for users around the world who have healthcare problems in particular and users across the world who want to search for information about healthcare and advice for avoiding health issues. Practical healthcare news, advice, and answers are provided to users who need fast and reliable information about health issues.

## The objectives and scope

The application provides main functions to help users find, search, and explore information related to healthcare problems and advice. This thesis will be focused on the problem of searching and providing reliable answers to the medical industry. The scope of the study is: researching the problem of searching and finding health-related information online and solving the problem by implementing a chatbot with friendly UI/UX and handy features.

## Research Methods:

- Requirements analysis: study the problem related to chatbots, especially in the medical field on the network, research papers, then analyze the function, and describe the requirements to build and train the chatbot.

- Data collection: Collect questions, answers, and relevant information about the medical problem to train the chatbot.

- Design: UI/UX design; Model analysis and design model architecture.

- Implementation: Using Python, and Google Colab for training the chatbot and then deploying the chatbot to Facebook messager with Python, and Flask using REST API.

## Result

- Using acquired knowledge of analysis, research, and information gathering, build an application with medical chatbot Python, and Machine Learning.

Thesis outline

**Chapter 1**: Introduction

**Chapter 2**: Background

General information about the study and main functions of the system.

**Chapter 3**: Design and implementation

Introduction of UI/UX designs, models, and implementation, describing

technologies that will be used in the study.

**Chapter 4**: Conclusion and future work

# PROBLEM DESCRIPTION

## Detailed description of the problem

Medical chatbot provides users with functions such as answering questions related to healthcare and searching for advice about health issues in an easy, fast, and reliable way. In addition, the chatbot feature is integrated into the website to help students get answers related to actual practice accurately and quickly.

Medical chatbot application consists of 2 main parts: the chatbot and the UI application.

The application part helps users interact with the chatbot and supports collecting questions and answers to improve the effectiveness of the training chatbot. In addition, the application section also supports users to find questions, answers, and news related to practical healthcare problems. The application section will also support an administrator function to maintain and update the chatbot to improve its accuracy of the model over time.

## Background

### Python

Python is a high-level, versatile programming language known for its simplicity and readability. Created by Guido van Rossum and first released in 1991, Python has gained widespread popularity in various domains, including web development, data science, artificial intelligence, scientific computing, and more.

Python is one of the most popular programming languages for machine learning and artificial intelligence. Its versatility, extensive libraries, and active community support make it an ideal choice for developing machine-learning models and conducting data analysis. Here's an overview of how Python is used in the field of machine learning:

1. Libraries and Frameworks: Python has a rich ecosystem of libraries and frameworks that simplify machine learning tasks. Some of the most commonly used ones include:

- numpy: numpy is a fundamental library for numerical computing in Python. It provides support for multi-dimensional arrays and mathematical functions, making it essential for data manipulation.

- pandas: pandas is a data manipulation library that provides data structures like DataFrames and Series. It is used for data cleaning, transformation, and analysis.

- scikit-learn: scikit-learn is a popular machine learning library that offers a wide range of machine learning algorithms for classification, regression, clustering, dimensionality reduction, and more. It also provides tools for model selection and evaluation.

- TensorFlow: Developed by Google, TensorFlow is an open-source machine learning framework that's widely used for deep learning tasks. It allows you to build neural networks for various applications, including image recognition, natural language processing, and reinforcement learning.

- PyTorch: PyTorch is another deep learning framework that has gained popularity for its flexibility and dynamic computation graph. It is known for its ease of use and is commonly used in research and development of neural network models.

- Keras: Keras is a high-level neural networks API that runs on top of other deep learning frameworks like TensorFlow and Theano. It simplifies the process of building and training neural networks.

2. Data Preprocessing: Python libraries like NumPy and pandas are instrumental for data preprocessing. You can clean and prepare your data, handle missing values, perform feature engineering, and create datasets suitable for machine learning tasks.

3. Visualization: Libraries like Matplotlib and Seaborn allow you to visualize your data, which is crucial for understanding patterns and relationships in your datasets. Visualization aids in data exploration and model evaluation.

4. Model Building and Training: Python's machine learning libraries provide a straightforward way to build, train, and evaluate machine learning models. You can experiment with various algorithms and techniques to find the best model for your specific problem.

5. Model Evaluation: Scikit-learn offers tools for evaluating machine learning models using metrics like accuracy, precision, recall, F1-score, and ROC curves. Cross-validation techniques help assess a model's generalization performance.

6. Deployment: After training a machine learning model, you can deploy it in a production environment. Python allows you to create web services, and RESTful APIs, or integrate models into applications using frameworks like Flask or Django.

7. Community and Resources: Python's machine learning community is vast, with a wealth of tutorials, documentation, and online courses available. Platforms like Kaggle provide datasets and competitions for practicing and honing your machine-learning skills.

8. Research: Python is commonly used for machine learning research, thanks to the availability of powerful libraries like TensorFlow and PyTorch. Researchers can experiment with cutting-edge techniques and algorithms.

Python's role in machine learning continues to evolve, and it remains at the forefront of AI and data science. Whether you're a beginner looking to get started or an experienced practitioner, Python's ecosystem and community support make it a valuable tool for tackling a wide range of machine-learning challenges.

### Tensorflow

TensorFlow is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

TensorFlow can run on multiple CPUs and GPUs (with optional CUDA and SYCL extensions for general-purpose computing on graphics processing units). TensorFlow is available on 64-bit Linux, macOS, Windows, and mobile computing platforms including Android and iOS.

Its flexible architecture allows for the easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices.

TensorFlow computations are expressed as stateful dataflow graphs. The name TensorFlow derives from the operations that such neural networks perform on multidimensional data arrays, which are referred to as tensors.

### Deep learning

Deep learning is the branch of machine learning which is based on artificial neural network architecture. An artificial neural network or ANN uses layers of interconnected nodes called neurons that work together to process and learn from the input data.

In a fully connected Deep neural network, there is an input layer and one or more hidden layers connected one after the other. Each neuron receives input from the previous layer neurons or the input layer. The output of one neuron becomes the input to other neurons in the next layer of the network, and this process continues until the final layer produces the output of the network. The layers of the neural network transform the input data through a series of nonlinear transformations, allowing the network to learn complex representations of the input data.

Deep learning can be used for supervised, unsupervised as well as reinforcement machine learning. It uses a variety of ways to process these.

### Chatbot

A chatbot (conversational agent (CA), dialogue system) is a computer software that acts as an interface between human users and a software application, using spoken or written natural language as the primary means of communication.

Chatbot components

The functioning of a chatbot involves several key components:

Natural Language Processing (NLP): Natural language understanding (NLU) is the first core component of the conversational agents which is responding by providing a semantic representation for user utterance [6] such as in the form of logic or class’s intent, extracting the “meaning” of an utterance. NLP is the technology that enables chatbots to understand and interpret human language.

Dialog Management: Dialogue Manager is The second core component in any chatbot and we can differentiate the chatbots through this component which has many parts that can be improved or adding some parts in the future if will be discovered that it will serve the DM.

Knowledge Base or Backend Integration: Many chatbots are connected to databases, APIs, or other systems to access information or perform actions. For example, a customer support chatbot might access a database of FAQs or connect to a CRM system to look up customer information.

User Interface: This is the medium through which users interact with the chatbot. It could be a chat window on a website, a messaging app like WhatsApp, or a voice-activated device like Amazon Echo.

Machine Learning (Optional): Some advanced chatbots incorporate machine learning algorithms. These bots can learn from user interactions to improve their responses over time. They become more effective as they gather more data and refine their understanding of user queries.

Types of chatbot

There are different types of chatbots based on their capabilities and functionalities:

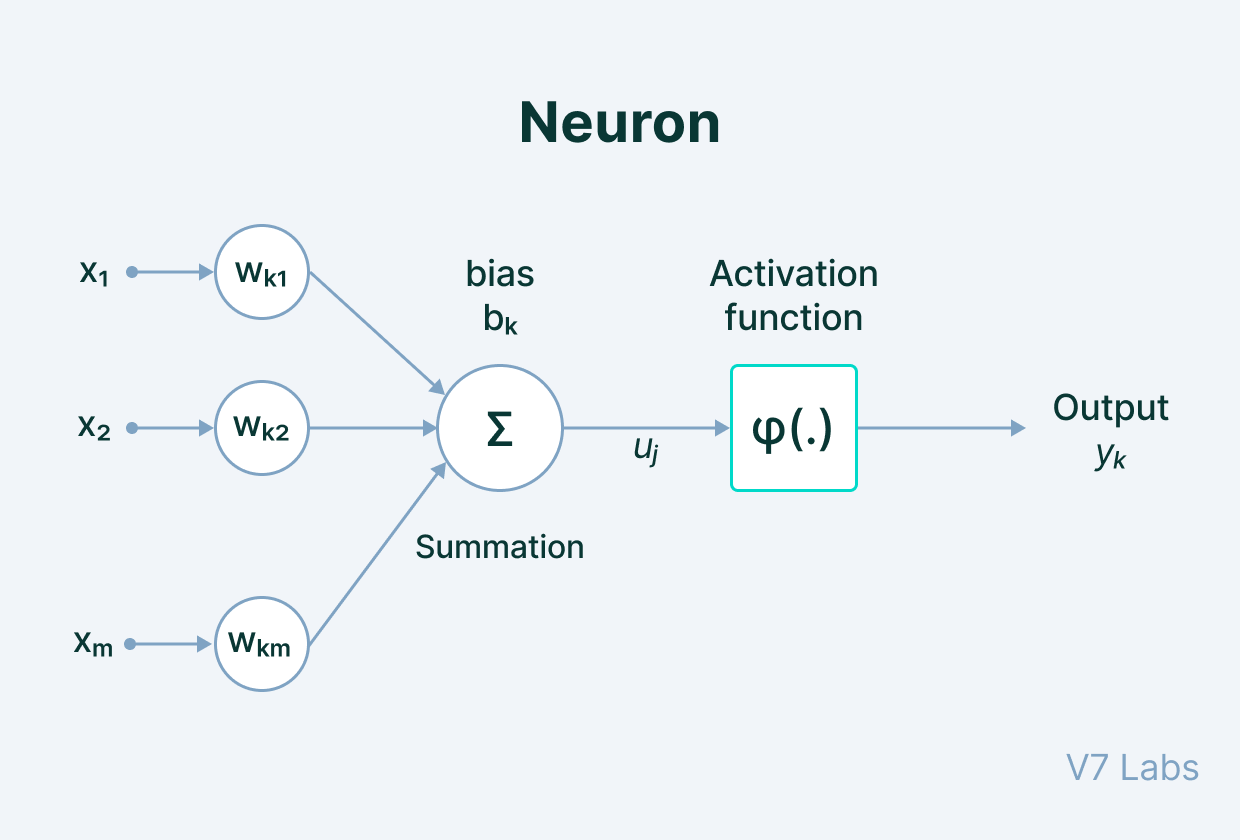
Rule-Based Chatbots: These chatbots follow predefined rules and patterns. They provide responses based on keywords and phrases. Rule-based chatbots are typically used for straightforward and specific tasks.

AI-Powered Chatbots: These chatbots leverage artificial intelligence and machine learning to provide more dynamic and context-aware responses. They can handle more complex and open-ended conversations.

Voice Assistants: Voice-activated chatbots, like Siri, Google Assistant, or Alexa, are designed to respond to spoken language inputs. They use speech recognition technology to understand and fulfill user requests.

### Neuron network

The inspiration for a Neural Network (NN) originates from the human brain, where biological neurons (nerve cells) respond to the activation of other neurons they are connected to. At a very simple level, neurons in the brain take electrical inputs that are then channeled to outputs as shown below.



Input: It is the set of features that are fed into the model for the learning process. For example, the input in object detection can be an array of pixel values pertaining to an image.

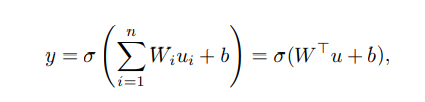
Weight: Its main function is to give importance to those features that contribute more towards the learning. It does so by introducing scalar multiplication between the input value and the weight matrix. For example, a negative word would impact the decision of the sentiment analysis model more than a pair of neutral words.

Transfer function: The job of the transfer function is to combine multiple inputs into one output value so that the activation function can be applied. It is done by a simple summation of all the inputs to the transfer function.

Activation Function: It introduces non-linearity in the working of perceptrons to consider varying linearity with the inputs. Without this, the output would just be a linear combination of input values and would not be able to introduce non-linearity in the network.

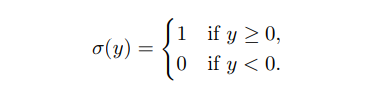
Bias: The role of bias is to shift the value produced by the activation function. Its role is similar to the role of a constant in a linear function.

For a single neuron/node with input u ∈ R n, a mathematical model, named the perceptron, can be described as

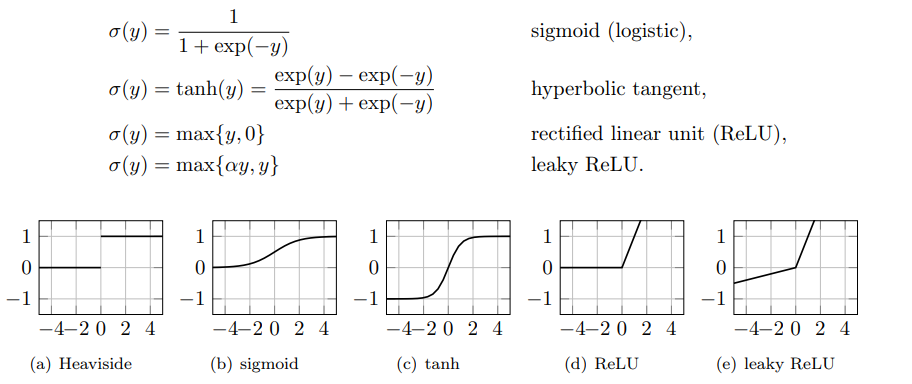


where y is the activation of the neuron/node, Wi are the weights and b is the bias.

The function σ : R → R is called activation function. Originally, it was proposed to choose the Heaviside function as activation function to model whether a neuron fires or not

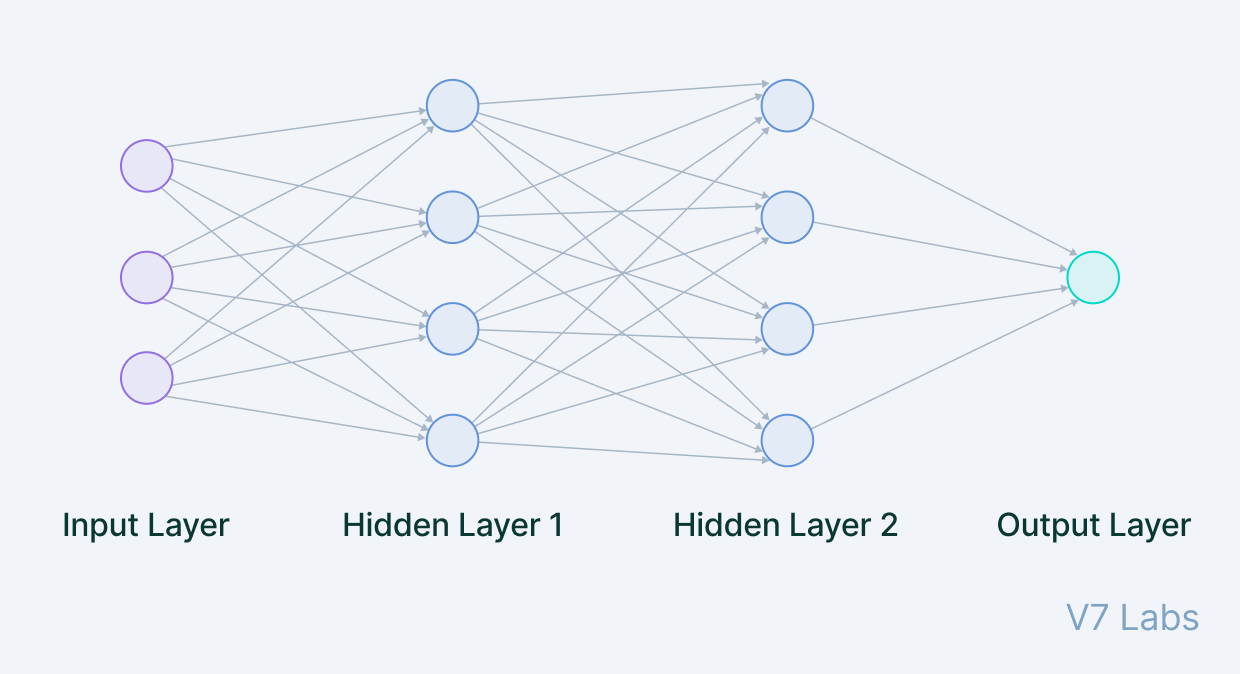


Popular activation functions are



When multiple neurons are stacked together in a row, they constitute a layer, and multiple layers piled next to each other are called a multi-layer neural network.

ANNs consist of interconnected artificial neurons or nodes organized in layers, each layer serving a specific purpose in the network. Here's an overview of the architecture of artificial neural networks:



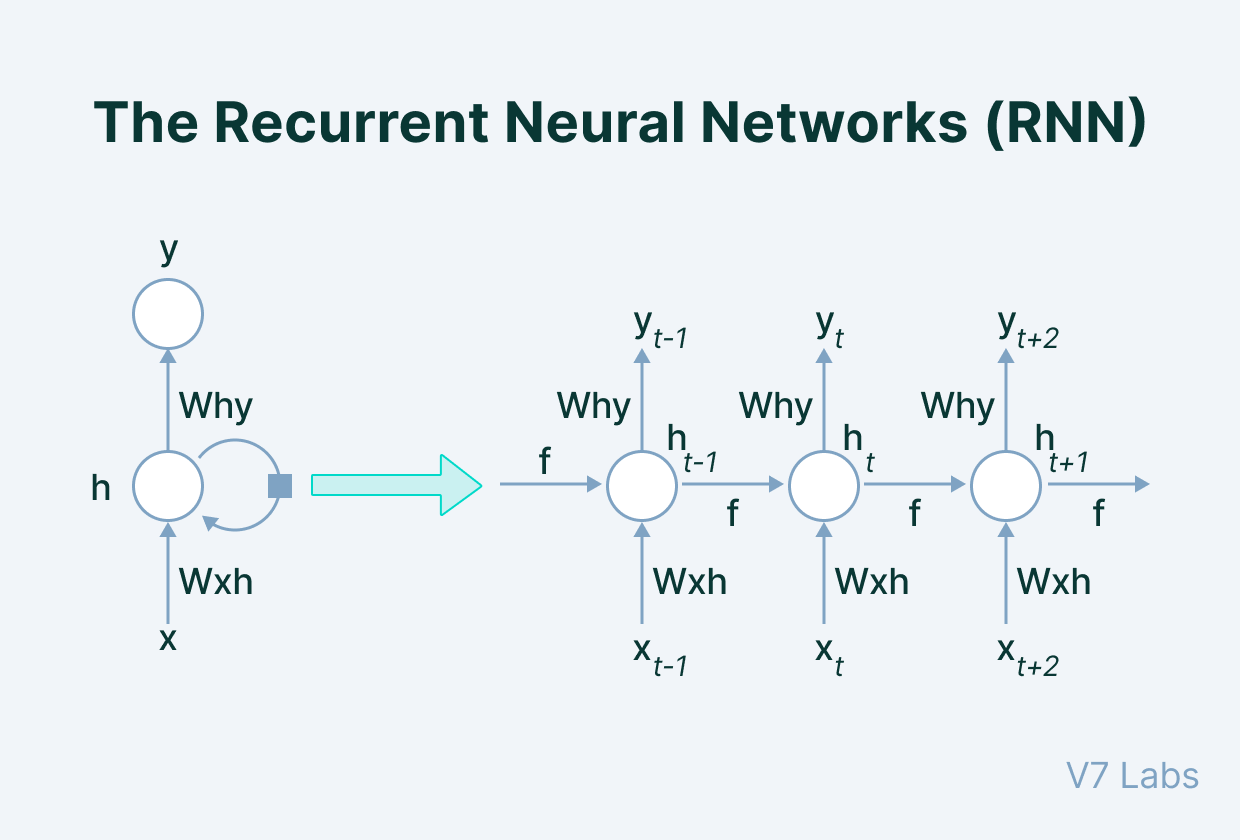
Input Layer: The input layer is the first layer of the neural network. Neurons in this layer receive input data, which could be features from a dataset or raw sensor data, depending on the application. The number of neurons in the input layer is determined by the dimensionality of the input data.

Hidden Layers: Between the input and output layers, there can be one or more hidden layers. Each hidden layer consists of multiple neurons. Neurons in the hidden layers process the input data through weighted connections and activation functions. The number of hidden layers and neurons in each layer is a hyperparameter that can be adjusted to optimize network performance.

Output Layer: The output layer takes input from preceding hidden layers and comes to a final prediction based on the model’s learnings. It is the most important layer where we get the final result. In the case of classification/regression models, the output layer generally has a single node. However, it is completely problem-specific and dependent on the way the model was built.

Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are a type of neural network architecture which is mainly used to detect patterns in a sequence of data. What differentiates Recurrent Neural Networks from Feedforward Neural Networks also known as Multi-Layer Perceptrons (MLPs) is how information gets passed through the network. While Feedforward Networks pass information through the network without cycles, the RNN has cycles and transmits information back into itself. This enables them to extend the functionality of Feedforward Networks to also take into account previous inputs and not only the current input. Recurrent Neural Networks have the power to remember what it has learned in the past and apply it in future predictions.



The input is in the form of sequential data that is fed into the RNN, which has a hidden internal state that gets updated every time it reads the following sequence of data in the input. The internal hidden state will be fed back to the model. The RNN produces some output at every timestamp.

We can describe this process of passing information from the previous iteration to the hidden layer as below:

Where:

: The hidden state at time step t

: The input at time step t

: The weight matrix

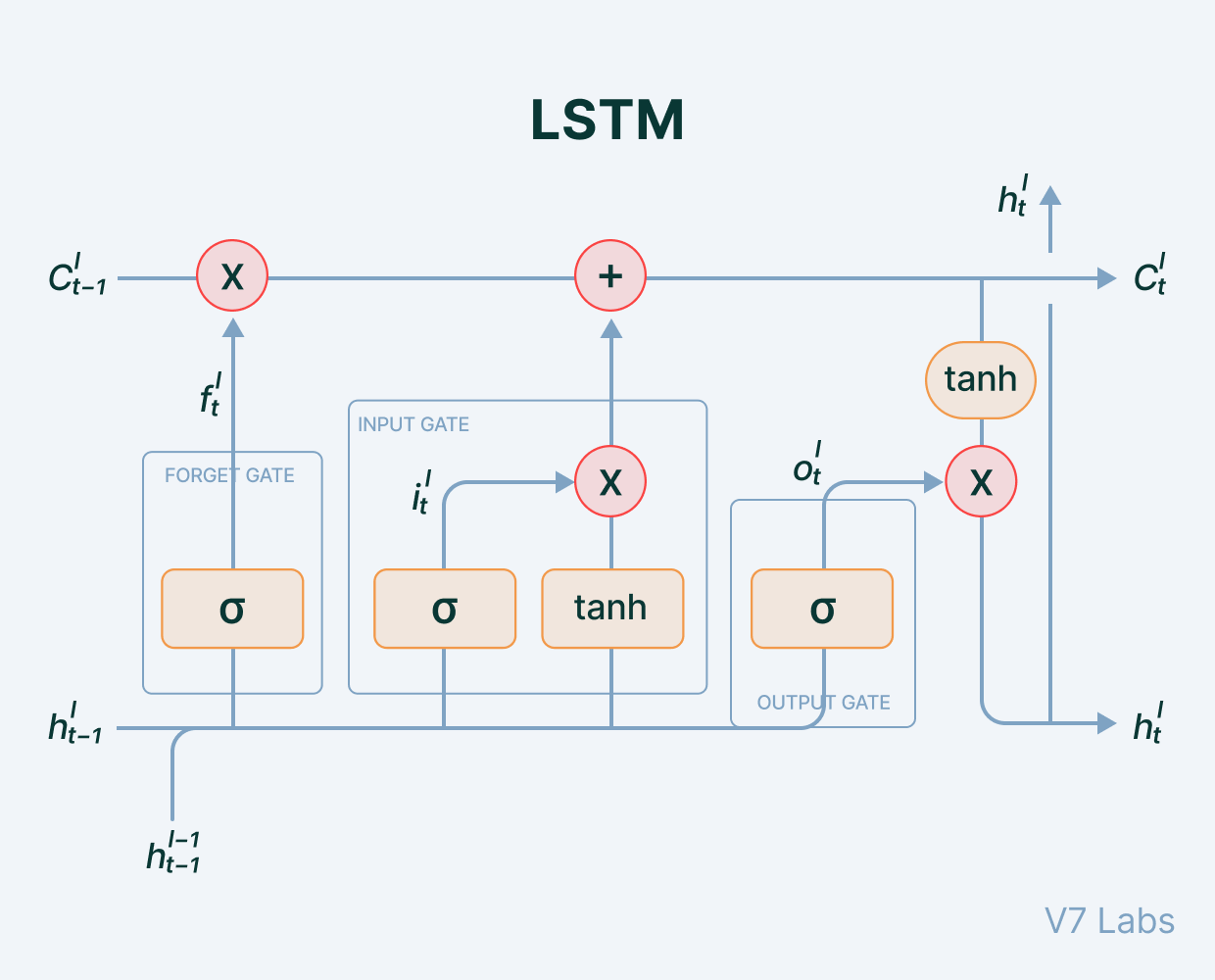
: The hidden-state-to-hidden-state matrix

: The bias

: The activation function

The Long Short Term Memory Network (LSTM)

Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies. In RNN each of our predictions looked only one timestamp back, and it has a very short-term memory. It doesn't use any information from further back. To rectify this, we can take our Recurrent Neural Networks structure and expand it by adding some more pieces to it. The critical part that we add to this Recurrent Neural Networks is memory. We want it to be able to remember what happened many timestamps ago. To achieve this, we need to add extra structures called gates to the artificial neural network structure.



Cell state: the key to LSTMs, the cell state is kind of like a conveyor belt. It runs straight down the entire chain, with only some minor linear interactions. It corresponds to the long-term memory content of the network.

The LSTM does have the ability to remove or add information to the cell state, carefully regulated by structures called gates. Gates are a way to optionally let information through. They are composed out of a sigmoid neural net layer and a pointwise multiplication operation. The sigmoid layer outputs numbers between zero and one, describing how much of each component should be let through.

Forget Gate: Some information in the cell state is no longer needed and is erased. The gate receives two inputs, x\_t (current timestamp input) and h\_t-1 (previous cell state), multiplied with the relevant weight matrices before bias is added. The result is sent into an activation function, which outputs a binary value that decides whether the information is retained or forgotten.

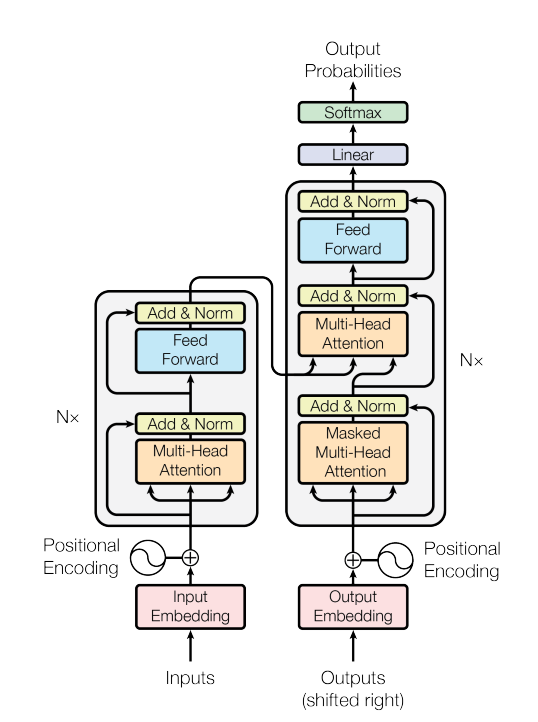
Input gate: It decides what piece of new information is to be added to the cell state. It is similar to the forget gate using the current timestamp input and previous cell state with the only difference of multiplying with a different set of weights.

Output gate: The output gate's job is to extract meaningful information from the current cell state and provide it as an output.

### Transformer

The Transformer was proposed in the paper “Attention is All You Need”. The Transformer models have an encoder-decoder structure. The encoder maps an input sequence of symbol representations to a sequence of continuous representations. The decoder then generates an output sequence of symbols one element at a time. At each step the model is auto-regressive consuming the previously generated symbols as additional input when generating the next.

The Transformer using stacked self-attention and point-wise, fully connected layers for both the encoder and decoder, shown in below.



#### Encoder

The encoder is composed of a stack of N identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, positionwise fully connected feed-forward network. We employ a residual connection around each of the two sub-layers, followed by layer normalization. That is, the output of each sub-layer is LayerNorm(x + Sublayer(x)), where Sublayer(x) is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension .

Embedding

An embedding is a mapping of a discrete, categorical variable to a vector of continuous numbers. In the context of neural networks, embeddings are low-dimensional, learned continuous vector representations of discrete variables. In the Transformer model, the embedding layer is used to represent the input into a low-dimensional vector using word embedding.

Positional Encoding

Since the model contains no recurrence and no convolution, in order for the model to make use of the order of the sequence, the model must be injected some information about the relative or absolute position of the tokens in the sequence. To this end, the "positional encodings" are added to the input embeddings at the bottoms of the encoder and decoder stacks. The positional encodings have the same dimension as the embeddings, so that the two can be summed. There are many choices of positional encodings, learned and fixed.

The sine and cosine functions are used to calculate the positional encoding of different frequencies:

where *pos* is the position and *i* is the dimension. That is, each dimension of the positional encoding corresponds to a sinusoid. The wavelengths form a geometric progression from 2π to 10000 · 2π.

Attention

Attention allows Transformer to look at other positions in the input sequence for clues that can help lead to a better encoding for this word. An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

Steps to calculate the attention:

Step 1: Creating three vectors from each of the encoder’s input vectors (in this case, the embedding of each word). So for each word, we create a Q vector (query vector), a K vector (key vector), and a V vector (value vector). These vectors are created by multiplying the embedding by three matrices that we trained during the training process.

Step 2: Calculating the score (the score determines how much focus to place on other parts of the input sentence as we encode a word at a certain position) by taking the dot product of the query vector with the key vector of the respective word.

Step 3: Divide the scores by the square root of the dimension of the key vectors (help stabling the gradients), then pass the result through a softmax operation. Softmax normalizes the scores so they’re all positive and add up to 1.

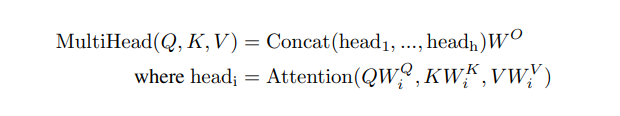
Step 4: Multiply each value vector by the softmax score to keep intact the values of the word(s) and drown-out irrelevant words.

Step 5: Sum up the weighted value vectors. This produces the output of the self-attention layer

Multi-Head Attention

Instead of performing a single attention function with dmodel-dimensional keys, values and queries, we found it beneficial to linearly project the queries, keys and values h times with different, learned linear projections to dk, dk and dv dimensions, respectively. On each of these projected versions of queries, keys and values we then perform the attention function in parallel, yielding dv-dimensional

output values. These are concatenated and once again projected, resulting in the final values, as depicted in Figure 2. Multi-head attention allows the model to jointly attend to information from different representation subspaces at different positions. With a single attention head, averaging inhibits this



Where the projections are parameter matrices W Q i ∈ R dmodel×dk , W K i ∈ R dmodel×dk , WV i ∈ R dmodel×dv and WO ∈ R hdv×dmodel . In this work we employ h = 8 parallel attention layers, or heads. For each of these we use dk = dv = dmodel/h = 64. Due to the reduced dimension of each head, the total computational cost is similar to that of single-head attention with full dimensionality

The Residuals

The residual connection is employed around each sub-layer (self-attention, ffnn) in each encoder and decoder, and is followed by a layer-normalization step.

Residual connection provides another path for data to reach latter parts of the neural network by skipping some layers. This help solving the problems such as exploding gradients and vanishing gradients when training the model.

Feed Forward

Each of the layers in our encoder and decoder contains a fully connected feed-forward network, which is applied to each position separately and identically. This consists of two linear transformations with a ReLU activation in between.

While the linear transformations are the same across different positions, they use different parameters from layer to layer.

#### Decoder

The decoder is also composed of a stack of N identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. The residual connections are also employed around each of the sub-layers, followed by layer normalization. The self-attention sub-layer in the decoder stack was modified to prevent positions from attending to subsequent positions. This masking, combined with fact that the output embeddings are offset by one position, ensures that the predictions for position *i* can depend only on the known outputs at positions less than *i*.

The components of the decoder and the encoder are pretty similar but have some different. The decoder start by processing the input sequence. The output of the top encoder is then transformed into a set of attention vectors K and V. These are to be used by each decoder in its “encoder-decoder attention” layer which helps the decoder focus on appropriate places in the input sequence.

Masked Multi-Head Attention

The embedding vectors are pass through the self-attention block, where attention vectors are generated for every word to represent how much each word is related to every word in the same sentence. Masking will be used to hide the next word so that, at first, it will predict the next word itself using previous results without knowing the real translated word. Only the previous word of the sentence are used for learning purposes. So, while performing parallelization with the matrix operation, the matrix will mask the words appearing later by transforming them into zeroes so that the attention network can’t use them.

Encoder-decoder attention block

The resulting attention vectors from the previous layer and the vectors from the encoder block are passed into another multi-head attention block. This is where the results from the encoder block also come into place. The output of this block is attention vectors for every word in the sentences. Each vector represents the relationship with other words in both sentences.

Linear and Softmax Layer

The Linear layer is a simple fully connected neural network that projects the vector produced by the stack of decoders, into a much, much larger vector called a logits vector. The softmax layer then turns those scores into probabilities (all positive, all add up to 1.0). The cell with the highest probability is chosen, and the word associated with it is produced as the output for this time step.

### Tokenizer

Tokenizing a text is splitting it into words or subwords, which then are converted to ids through a look-up table. Converting words or subwords to ids is straightforward, so we will focus on splitting a text into words or subwords (i.e. tokenizing a text)

Subword-based tokenization

Subword-based tokenization is a solution between word and character-based tokenization. The main idea is to solve the issues faced by word-based tokenization (very large vocabulary size, large number of OOV tokens, and different meanings of very similar words) and character-based tokenization (very long sequences and less meaningful individual tokens).

The subword-based tokenization algorithms do not split the frequently used words into smaller subwords. It rather splits the rare words into smaller meaningful subwords. For example, “boy” is not split but “boys” is split into “boy” and “s”. This helps the model learn that the word “boys” is formed using the word “boy” with slightly different meanings but the same root word.

Some of the popular subword tokenization algorithms are WordPiece, Byte-Pair Encoding (BPE), Unigram, and SentencePiece.

Byte-Pair Encoding (BPE)

BPE is a simple form of data compression algorithm in which the most common pair of consecutive bytes of data is replaced with a byte that does not occur in that data. BPE relies on a pre-tokenizer that splits the training data into words. Pretokenization can be as simple as space tokenization.

After pre-tokenization, a set of unique words has been created and the frequency with which each word occurred in the training data has been determined. Next, BPE creates a base vocabulary consisting of all symbols that occur in the set of unique words and learns to merge rules to form a new symbol from two symbols of the base vocabulary. It does so until the vocabulary has attained the desired vocabulary size. Note that the desired vocabulary size is a hyperparameter to define before training the tokenizer.

Encoding and Decoding

Let us now see how we will decode our example. To decode, we have to simply concatenate all the tokens together to get the whole word. For example, the encoded sequence [“the</w>”, “high”, “est</w>”, “range</w>”, “in</w>”, “Seattle</w>”], we will be decoded as [“the”, “highest”, “range”, “in”, “Seattle”] and not as [“the”, “high”, “estrange”, “in”, “Seattle”]. Notice the presence of the “</w>” token in “est”.

For encoding the new data, the process is again simple. However, encoding in itself is computationally expensive. Suppose the sequence of words is [“the</w>”, “highest</w>”, “range</w>”, “in</w>”, “Seattle</w>”]. We will iterate through all the tokens we found in our corpus — longest to the shortest and try to replace substrings in our given sequence of words using these tokens. Eventually, we will iterate through all the tokens and our substrings will be replaced with tokens already present in our token list. If a few substrings are left (for words our model did not see in training), we will replace them with unknown tokens.

In general, the vocabulary size is big but still, there is a possibility of an unknown word. In practice, we save the pre-tokenized words in a dictionary. For unknown (new) words, we apply the above-stated encoding method to tokenize the new word and add the tokenization of the new word to our dictionary for future reference. This helps us build our vocabulary even stronger for the future.

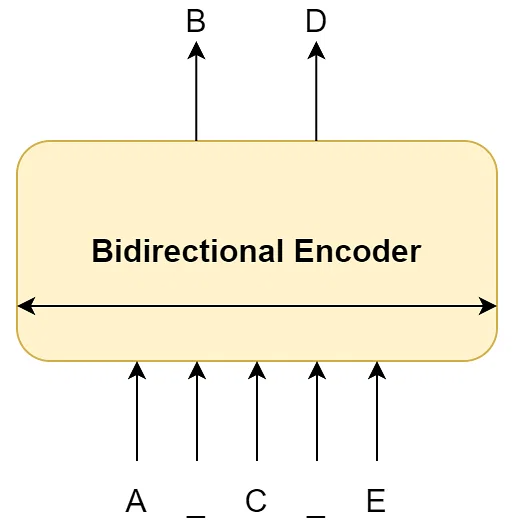
### BART model

BART is a denoising autoencoder that maps a corrupted document to the original document it was derived from. It is implemented as a sequence-to-sequence model with a bidirectional encoder over corrupted text and a left-to-right autoregressive decoder. For pre-training, we optimize the negative log-likelihood of the original document.

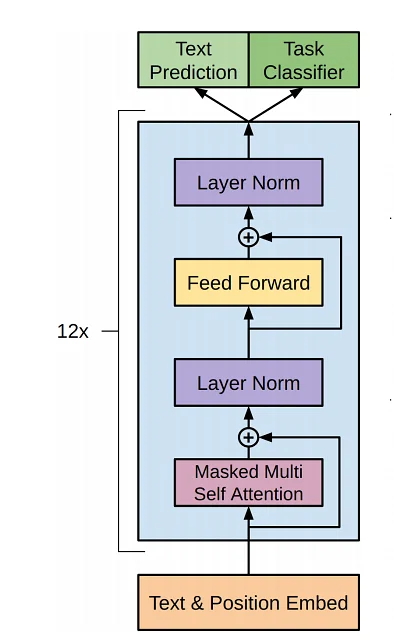
BART uses the standard sequence-to-sequence Transformer architecture from the original Transformer of Google, except, following GPT, BART modifies ReLU activation functions to GeLUs and initializes parameters from N (0, 0.02). For the base model, BART uses 6 layers in the encoder and decoder, and for the large model, BART uses 12 layers in each. The architecture is closely related to that used in BERT, with the following differences:

* Each layer of the decoder additionally performs cross-attention over the final hidden layer of the encoder (as in the transformer sequence-to-sequence model)
* BERT uses an additional feed-forward network before word prediction, which BART does not. In total, BART contains roughly 10% more parameters than the equivalently sized BERT model.

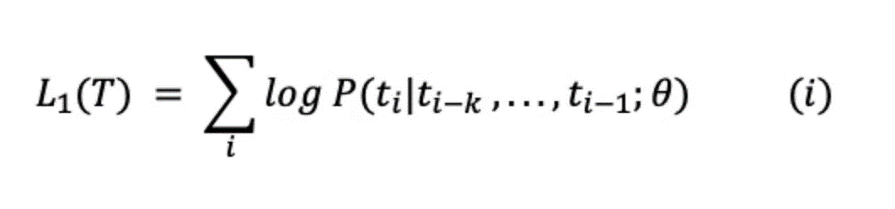
For every text sequence in its input, the BERT encoder outputs an embedding vector for each token in the sequence as well as an additional vector containing sentence-level information. In this way, the decoder can learn for both token and sentence-level tasks making it a robust starting point for any future fine-tuning tasks.



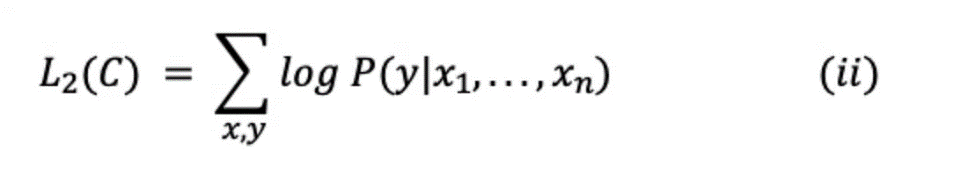
Once we get the token and sentence-level representation of an input text sequence, a decoder needs to interpret these to map with the output target. However, by using a similarly designed decoder, tasks such as next-sentence prediction or token prediction might perform poorly since the model relies on a more comprehensive input prompt. In these cases, we need model architectures that can be trained to generate the next word by only looking at the previous words in the sequence. Hence, a causal or autoregressive model that looks only at the past data to predict the future comes in handy.



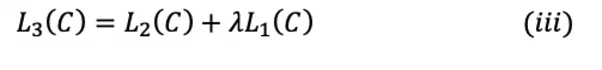
First, the model is pre-trained on tokens “t” looking back to “k” tokens in the past to compute the current token. This is done unsupervised on a vast text corpus to allow the model to “learn the language.”



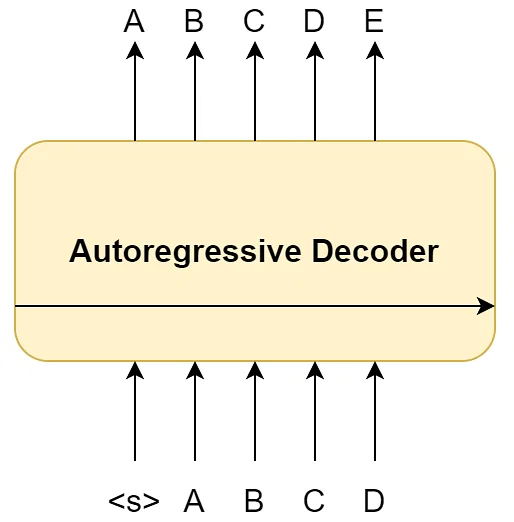
Next, to make the model robust on a specific task, it is fine-tuned in a supervised manner to maximize the likelihood of label “y” given feature vectors x1…xn.



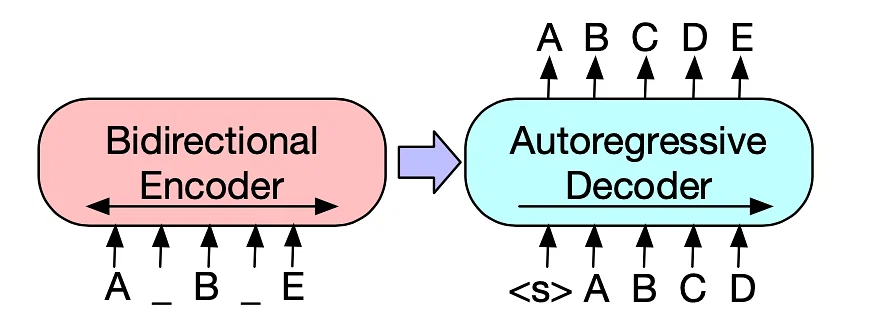
Combining 1 and 2, we get the objective in 3. Lambda represents a learned weight parameter to control the influence of language modeling.



The below image shows how the autoregressive decoder processes its input.



Although we separate the decoder from an encoder, the input to the decoder would still be a learned representation (or embedding) of the original text sequence. Thus, BART attaches the bi-directional encoder to the autoregressive decoder to create a denoising auto-encoder architecture. Based on these two components, the final BART model would look something like this:



In the above figure, the input sequence is a masked (or noisy) version of [ABCDE] transformed into [A(MASK)B(MASK)E]. The encoder looks at the entire sequence and learns high-dimensional representations with bi-directional information. The decoder takes these thought vectors and regressively predicts the next token. Learning occurs by computing and optimizing the negative log-likelihood as mapped with the target [ABCDE].

### BLEU Evaluation

BLEU, or the Bilingual Evaluation Understudy, is a metric for comparing a candidate translation to one or more reference translations. Although developed for translation, it can be used to evaluate text generated for different natural language processing tasks, such as paraphrasing and text summarization.

The BLEU score is not perfect, but it’s quick and inexpensive to calculate, language-independent, and, above all, correlates highly with human evaluation.

The BLEU score can be calculated as below:

With BP is the brevity penalty factor:

Where:

: the modified n-gram precisions measure how many n-grams in the reference sentence are reproduced by the candidate sentence.

: The number of unigrams, bigrams, trigrams, and four-grams (*i*=1,...,4) match their n-gram counterpart in the reference translations.

: The total number of unigrams in the candidate.

The BLEU metric ranges from 0 to 1. It is zero iff none of the n-substrings in the candidate is in reference. It is one iff every n-gram in the candidate that appears in reference, at least as many times as in the candidate.

# DESIGN AND IMPLEMENT OF THE CHATBOT

## Overview

The process for building the chatbot includes the following steps:

* Collecting dataset: research, crawl, and construct the dataset for training the model.
* Preprocessing: From the dataset obtained in the above step, proceed with eliminating special characters in sentences (!@#%&\*,..), remove multiple spaces, blank lines, remove meaningless words, short sentences (hmm, ag,..), duplicate sentences, etc. Then save the results into the .csv file.
* Training the model: Read the datasets obtained in the above step, then input them into the model for training purposes.
* Inferencing: Enter input data, and proceed to process that data similarly as a preprocessing step but does not have to be saved to a file. Then use the trained model to predict the answer, and print the results to the screen.
* Integrate to application: The trained model will be used to create an API for the applications to send the request containing the question and retrieve the answer back in the response. The answer will be extracted in the response to display on the UI for the users in various applications.

## Implementation

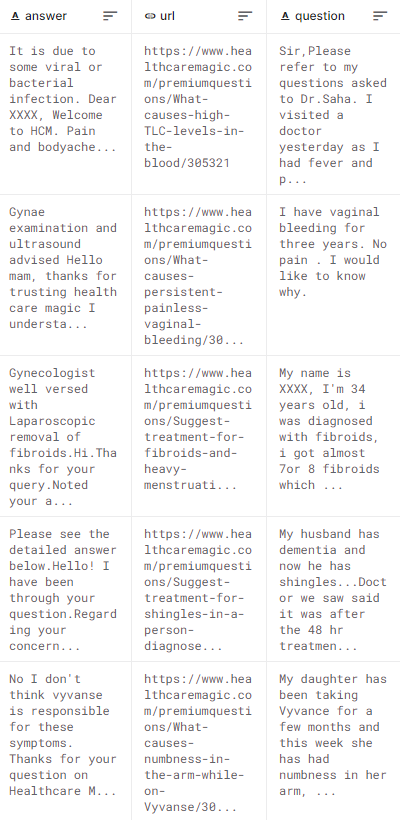
### Collecting dataset

To train the bot to answer questions about healthcare, a list of collected questions and answers needs to be analyzed and processed. Based on the analysis, the final data to train the chatbot is organized and defined in train, test, and validation dataset files. Once the data is finalized, the chatbot needs to be fed all the data with the corresponding responses. Then the chatbot created needs to be tested and trained to fine-tune the results.

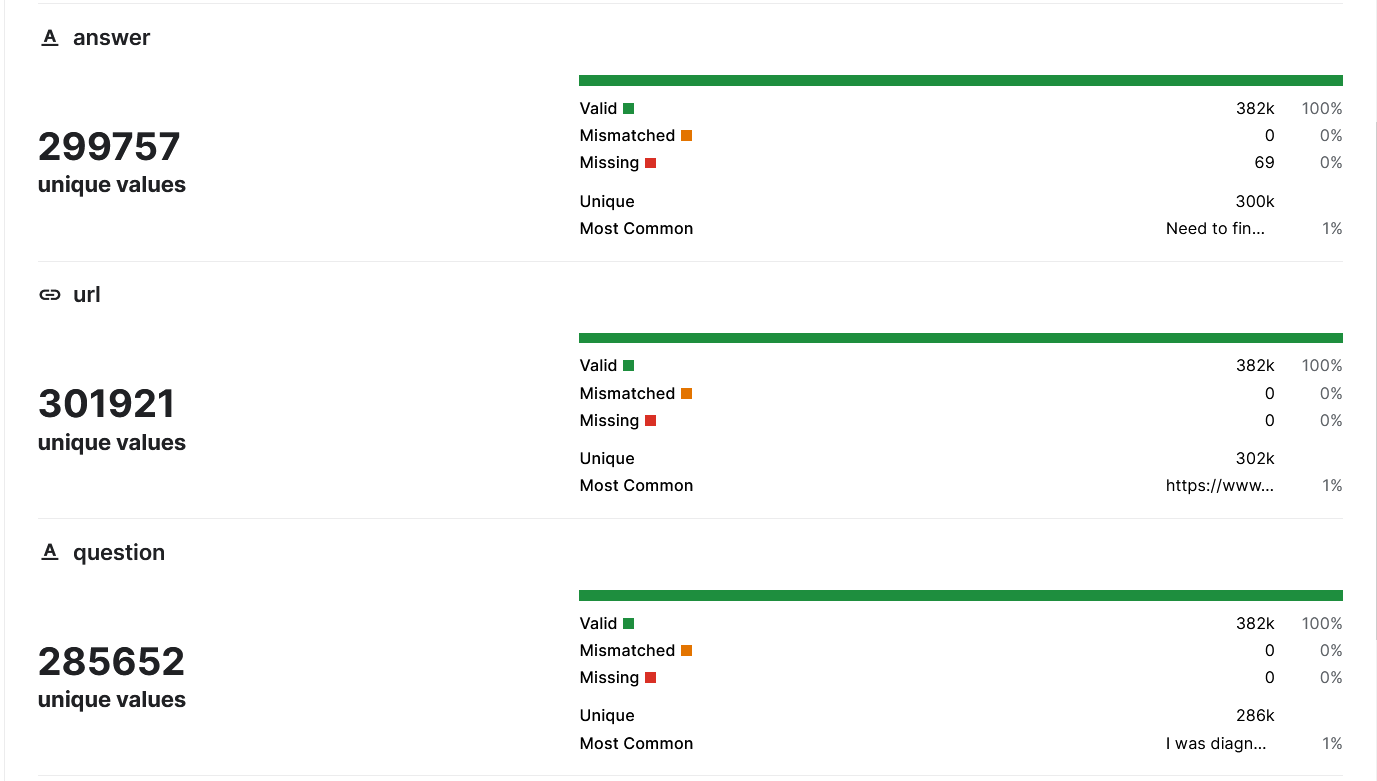
There are 3 datasets are collected and used for training the chatbot:

* The EHealthChatMini dataset is the dataset collected from crawling on popular healthcare websites (English) with more than 299,000 question-answer pairs related to healthcare.

Below are some examples from the dataset:



Also, details information about the dataset is shown below:

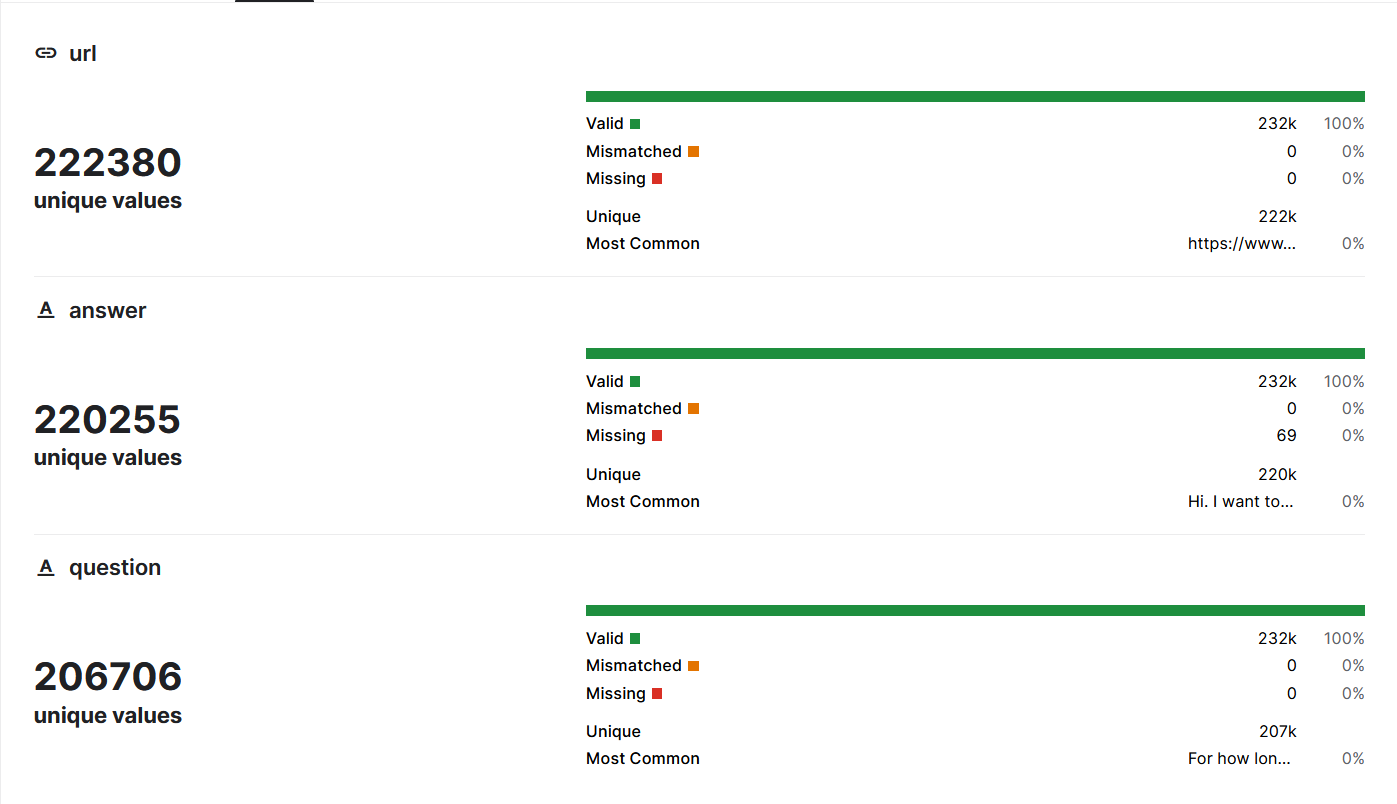


* The EHealthChat dataset is the dataset provided by Kaggle (English) with more than 222,000 question-answer pairs about healthcare problems.

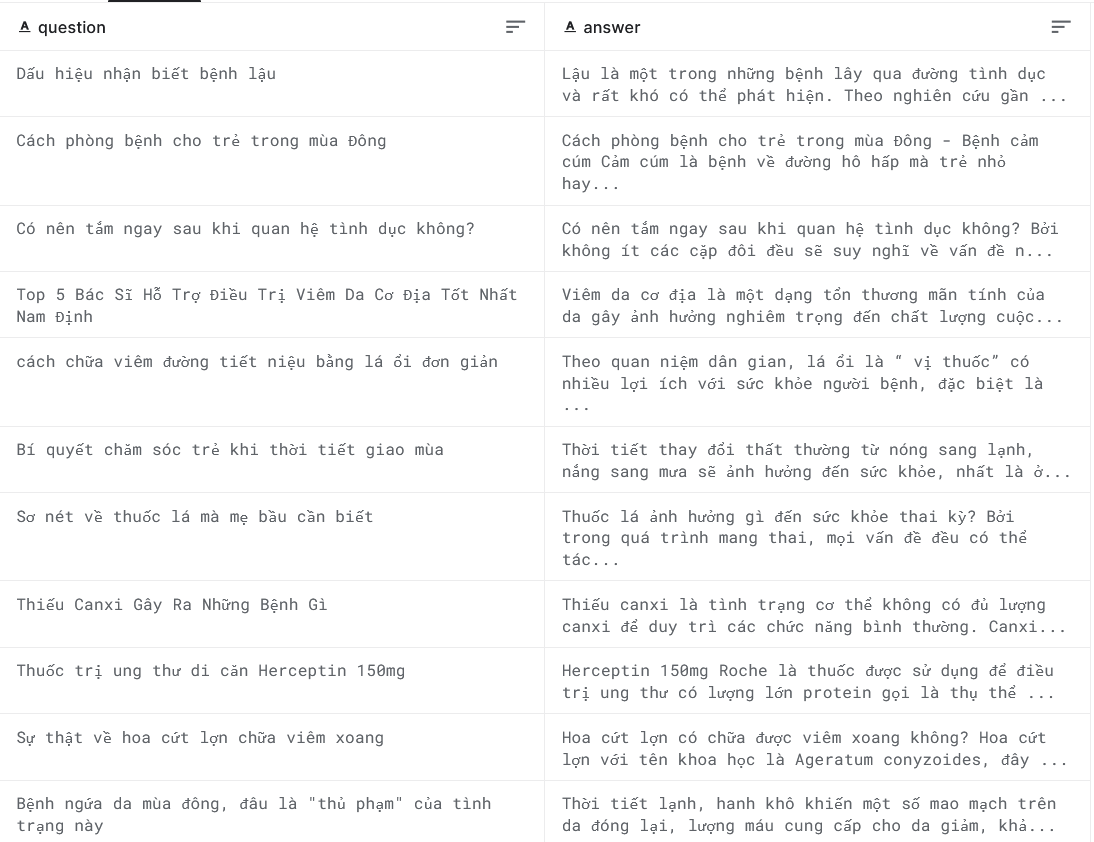
Below are some examples from the dataset:



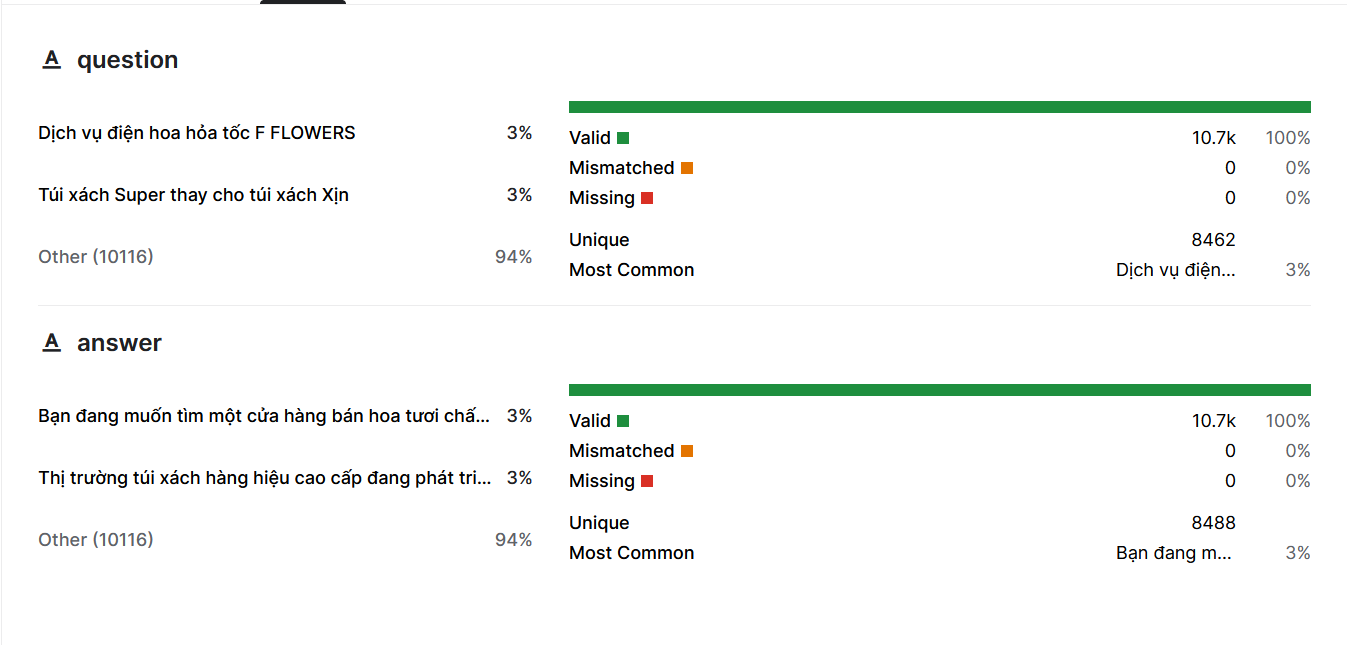
Also, details information about the dataset is shown below:



* The EHealthVNChat dataset is the dataset collected from crawling popular healthcare websites (Vietnamese) with more than 10,000 question-answer pairs related to healthcare.
* Below are some examples from the dataset:



Also, details information about the dataset is shown below:



### Preprocessing

The data was obtained in raw form, containing many special characters, along with many words abbreviations or words that have no meaning, .... Data cleaning steps are performed as follows:

* Remove all special characters in the sentence (@#$%^,..).
* Remove all sub, watermarked sentences.
* Remove multiple blank lines.
* Remove all sentences with lengths below 6.

After completing the preprocessing dataset, the processed datasets with be converted into CSV files for easy and suitable training of the model with datasets loader libraries.

### Training

The training process is implemented as described below:

The English model version is trained using two processed datasets as above, and the Vietnamese version uses the VN processed dataset for training purposes.

Firstly, the dataset will be converted into the batched dataset, where each batch contains N samples (forming a matrix from multiple vector input to speed up the training process), the input matrix will be passed into the tokenizer to convert into the token IDs (numerical value) and are passed to the embedding layer to convert into the vector and adding the position encoding to capture the position info inside the input. The different length vectors are padded into a fixed length with padding value (usually 0) to optimize compute matrix multiplication. Next, the data will be passed through the encoder and decoder block respectively, and finally flow through the linear and softmax layer to produce the probability of the next word for all the words in the dictionary. The softmax output will be used to compute the loss, and then compute the gradients for updating the model weights. After processing all the batches inside the dataset, the model has finished an epoch. The model will repeat this process for a fixed amount of time (called epochs). Finally, the model will be saved for inference later and integrated into other applications.

### Inferencing

After the model has been trained, it can be used to generate answers from user inputs. The steps to inferencing the model can be described below:

The user input will be pre-processed and tokenized into token IDs, then the token IDs will be fed into the model to generate the probability of the next token, the index of the token with the highest probability will be mapped into the token by the tokenizer decoding. The previous predicted token will be concat into the user input to form a new input and the process begins with the new input being the original input that has been concat with the predicted token. The process will continue until the max sequence length reaches or the end of string token is found.

## Integrate to application

### Facebook messager

There are 3 main steps to deploy the chatbot to the Facebook messager application as shown below:

* Create a server that listens to messages from Facebook (using Flask and Python)
* Define a function for sending messages back to users (using requests)
* Forward an HTTPS connection to the flask server (using ngrok or serveo)

#### The server

The first step is to create an HTTP server that listens to messages sent by Facebook, gets a response to that message, and eventually sends the response back to the user on Facebook. The Flask Python framework will be used to create this server. The basic idea is the following:

* Create a flask app that listens for messages sent to localhost:5000/webhook. When messages are sent on Facebook they will arrive as HTTP requests to this URL.
* The listen() function handles these HTTP requests and checks that they contain a valid Facebook message
* If the message is valid, the get\_response() function is called and the response is sent back to Facebook Messenger

In addition, the function verify\_webhook() needs to be implemented to handle the initial authentication between Facebook and the server

#### The response

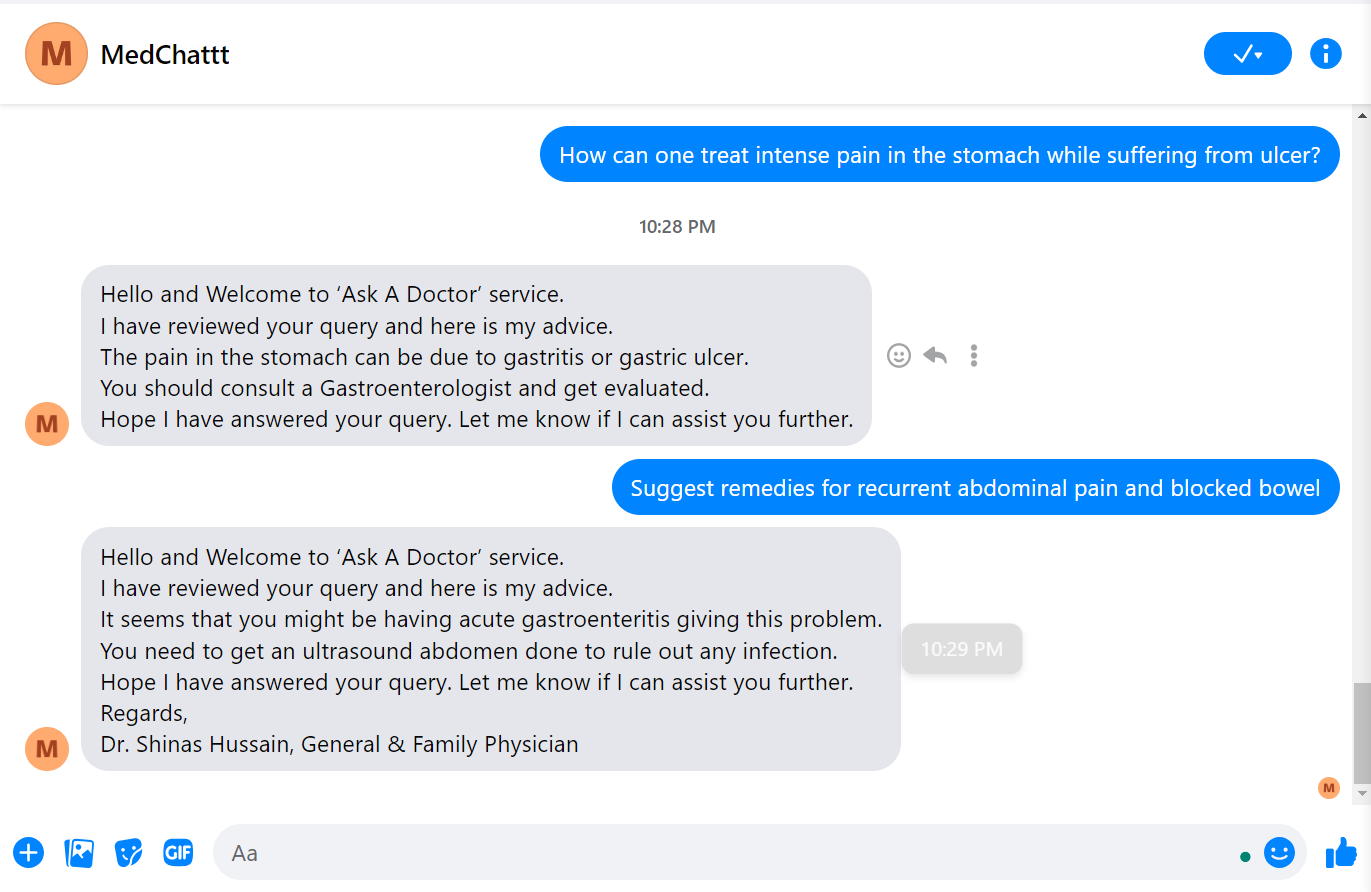
A function that sends a response back to Facebook Messenger using a Python library called requests. In particular, the method will get the response from the chatbot according to the message included in the request and send it back to Facebook with a POST request.

#### Expose a HTTPS endpoint

The ngrok or serveo server will be used to set up an HTTPS endpoint to get forwarded to the Flask server.

#### Create a Facebook app and page

The next step is to create an app and a page on Facebook. Once the page has been created, go to the Token Generation settings and select this page from the drop-down menu. Copy the Page Access Token into the placeholder for PAGE\_ACCESS\_TOKEN in the Flask server. After that, the server webhook will be registered on the Facebook developer's page. Finally, just start the chatbot server and the users can chat with the chatbot via the Facebook messager. Below is an example of the chatbot after integrated to Facebook Messager.



### Mobile app

There are 4 main steps to deploy the chatbot to an mobile application as described below:

* Create a server that listens to messages from the application (using Flask and Python)
* Define a function for sending messages back to users (using requests)
* Create an mobile application UI to interact with the chatbot server using Flutter
* Forward an HTTPS connection to the flask server (using ngrok or serveo)

#### The server

The first step is to create an HTTP server that listens to messages sent by the mobile application, gets a response to that message, and eventually sends the response back to the user on the mobile app. The Flask Python framework will be used to create this server. The basic idea is the following:

* Create a flask app that listens for messages sent to localhost:5000/webhook. When messages are sent on the mobile they will arrive as HTTP requests to this URL.
* The chat() function handles these HTTP requests and checks that they contain a valid message.
* If the message is valid, the get\_response() function is called and the response is sent back to application.

#### The response

A function that sends a response back to the mobile application using a Python library called requests. In particular, the method will get the response from the chatbot according to the message included in the request and send it back via a POST request.

#### Create an mobile application

1. **Sign up feature**

The application allow the user to create an account to help keep track of the data and the conversation with the chatbot. When users sign up, they have to enter their email address, username, and password. If there are some errors in the signing-up process, the system will show notifications. Otherwise, show a success notification to the user.

1. **Sign in feature**

Along the sign up feature, the users can also sign in with a signed-up email and password, and the system will authenticate, if the email and password are both valid and, in the database, the system will direct users to the conversation pages. Otherwise, the system will show an error message.

1. **Chat feature**

The chat feature allows users to ask and receive answers related to healthcare problems from the chatbot server. When users input a question, the chatbot will generate a response from the data it had trained before and send it to the users. Users can ask as much as they want and the conversation (the questions and answers) will be saved and the user can view it later if they wish.

#### Expose a HTTPS endpoint

The ngrok or serveo server will be used to set up an HTTPS endpoint to get forwarded the requests of the application to the Flask server.

### Web application

There are 4 main steps to deploy the chatbot to a web application:

* Create a server that listens to messages from the application (using Flask and Python)
* Define a function for sending messages back to users (using requests)
* Create a web application UI to interact with the chatbot server using VueJs, NodeJs
* Forward an HTTPS connection to the flask server (using ngrok or serveo)

#### The server

The first step is to create an HTTP server that listens to messages sent by the mobile application, gets a response to that message, and eventually sends the response back to the user on the mobile app. The Flask Python framework will be used to create this server. The basic idea is the following:

* Create a flask app that listens for messages sent to localhost:5000/webhook. When messages are sent on the mobile they will arrive as HTTP requests to this URL.
* The chat() function handles these HTTP requests and checks that they contain a valid message.
* If the message is valid, the get\_response() function is called and the response is sent back to application.

#### The response

A function that sends a response back to the web application using a Python library called requests. In particular, the method will get the response from the chatbot according to the message included in the request and send it back via a POST request.

#### Create an web application

1. **Sign up feature**

The application allow the user to create an account to help keep track of the data and the conversation with the chatbot. When users sign up, they have to enter their email address, username, and password. If there are some errors in the signing-up process, the system will show notifications. Otherwise, show a success notification to the user.

1. **Sign in feature**

Along the sign up feature, the users can also sign in with a signed-up email and password, and the system will authenticate, if the email and password are both valid and, in the database, the system will direct users to the conversation pages. Otherwise, the system will show an error message.

1. **Chat feature**

The chat feature allows users to ask and receive answers related to healthcare problems from the chatbot server. When users input a question, the chatbot will generate a response from the data it had trained before and send it to the users. Users can ask as much as they want and the conversation (the questions and answers) will be saved and the user can view it later if they wish.

#### Expose a HTTPS endpoint

The ngrok or serveo server will be used to set up an HTTPS endpoint to get forwarded the requests of the application to the Flask server.

# EVALUATION

## Result

The model is trained on Google Colab (with 12GB RAM and GPU T4) as well as on Kaggle (with 12GB RAM GPU P100) for 100 epochs, each epoch takes about 3 hours to complete.

Firstly, the chatbot was trained on the Medical Chat Dataset with more than 200,000 question-answer pairs about healthcare.

Next, the chatbot continues to train on the collected datasets crawled from healthcare websites with more than 100,000 question-answer pairs.

The chatbot was initially trained with different max sequence lengths (64, 128, 256, 512) on a relatively small dataset (about 10,000 question-answer pairs) to find the optimized hyper-parameters but still have good accuracy.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sequence length | 64 | 128 | 256 | 512 |
| Epochs | 100 | 100 | 100 | 100 |
| Dataset | 222,380 pairs | 222,380 pairs | 222,380 pairs | 222,380 pairs |
| Word dictionary |  |  |  |  |
| Time for 1 epoch | 45 minutes | 1 hour | 1 hour 15 mins | 2 hours |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sequence length | 64 | 128 | 256 | 512 |
| Epochs | 100 | 100 | 100 | 100 |
| Dataset | 299,757 pairs | 299,757 pairs | 299,757 pairs | 299,757 pairs |
| Word dictionary |  |  |  |  |
| Time for 1 epoch | 45 minutes | 1 hour | 1 hour 15 mins | 2 hours |

# CONCLUSION AND FUTURE WORK

## Conclusion

## Future work

# REFERENCES

# APPENDICES