



Höhere Technische
Bundeslehranstalt Weiz



DIPLOMARBEIT

Höhere Abteilung für Elektrotechnik

Untersuchung KI und praktische KI Anwendung

Diplomand

Riccardo Wolf

Klasse

5.BHET

Jahrgang

2023/24

Betreuer

Prof. DI Christoph Wurzinger

Projektpartner

Learn4D365 (Rene Gayer)

Abgabevermerk

03.04.2024

Unterschrift:

DIPLOMARBEIT

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche erkenntlich gemacht habe.

Weiz, am 03.04.2024, Riccardo Wolf:

.....

DIPLOMARBEIT

Kurzbeschreibung

Namen der Verfasser	Riccardo Wolf
Jahrgang	5.BHET
Schuljahr	2023/24
Thema der Diplomarbeit	Untersuchung KI und praktische KI Anwendung
Kooperationspartner	Learn4D365 (Rene Gayer)
Approbation (Datum / Unterschrift)	

Individuelle Themenstellung

Anhand eines Praxisversuches der Einbindung von KI in die vorhandene Website sollen die möglichen Anwendungen, sowie die eventuellen spezifischen Anpassungen welche für verschiedenste Unternehmen nötig sind, um erfolgreich ein KI-System für ihren eigenen Nutzen aufzusetzen untersucht werden. In dem begleiteten Praxisbeispiel dieser Diplomarbeit handelt es sich um die Einbindung eines LLM (Large-Language-Models) als Support-Bot welcher verschiedenste Fragen der Nutzer über den beinhalteten Lerninhalt beantworten soll. Das Ziel dieser Arbeit ist jedoch nicht nur die reine Umsetzung eines Support-Bots und dessen Training, sondern des Weiteren auch die allgemeine Untersuchung verschiedenster KI-Modelle und dessen Anwendung auf spezifische Unternehmen und deren Branchen.

Realisierung

Die Realisierung des Projektes erfolgte mit mehrfacher Rücksprache des Auftraggebers, durch laufenden Kontakt mit dem Projektpartner wurde eine fließende Entwicklung bewerkstelligt.

Ergebnisse

Durch mehrfache Absprache mit dem Kooperationspartner wurde ein proof-of-concept Prototype mithilfe von verschiedenster Technologien (PHP, Javascript, OpenAI-API) realisiert. Der entwickelte Prototype ist online auf eine vom Auftraggeber zur Verfügung gestellten Website (<https://www.saindy.io/>) verfügbar.

DIPLOMA THESIS

Abstract

Author(s)	Riccardo Wolf
Form	5.BHET
Academic Year	2023/24
Topic	Research AI and practical AI application
Co-operation Partner	Learn4D365 (Rene Gayer)
Approbation (Date / Signature)	

Individual Topic

Through a practical trial of integrating AI into the existing website, the possible applications, as well as the potential specific adjustments that are necessary for various companies to successfully set up an AI system for their own use, are to be examined. In the accompanying practical example of this thesis, it involves the integration of an LLM (Large-Language-Model) as a support bot which is intended to answer various questions of users about the included learning content. However, the goal of this work is not only the mere implementation of a support bot and its training, but also the general examination of various AI models and their application to specific companies and their industries.

Realization

The realization of the project was achieved over multiple consultations with the client. Through ongoing contact with the project partner, a smooth development was ensured.

Results

Through multiple discussions with the cooperation partner, a proof-of-concept prototype was realized using various technologies (PHP, Javascript, OpenAI API). The developed prototype is available online on a website provided by the client (<https://www.saindy.io/>).

DIPLOMA THESIS

Preface

Personal motivation

In our technological reliant world, AI is growing faster than ever. Since the release of the GPT LLM-Model series from OpenAI to the public, especially their service ChatGPT, there has been an absolutely massive influx in interest for AI and its applications. Since I personally started following AI ever since I discovered the text-davinci series way back in 2017, I figured it was only fitting my diploma thesis incorporates AI.

Special thanks

First and foremost I would like to thank Rene Gayer for allowing me to build his company this prototype and work together on such new and exciting technology. I would also like to thank my supervisor Christoph Wurzinger for accepting my thesis proposal.

Also a very special thanks to all the teachers that bestowed their knowledge upon me while I had the pleasure of educating myself at this facility.

Cooperation partners

Learn4D365 describes themselves as follows:

We are an online learning platform consisting of CRM and ERP specialists, where each individual contributes their expertise in Microsoft Dynamics. Furthermore, we engage with future-oriented technologies and products to possibly recommend new products or technologies to our customers today. Our mission is to pass on our practical know-how to companies and their employees in a practical manner. We speak the language of the participants and bring technical knowledge to a level where it can be used efficiently and productively. Through collaboration with the manufacturer, we bring knowledge from the source to users, professionals and IT experts.

Table of contents

Eidesstattliche Erklärung	1
Kurzbeschreibung	2
Abstract	4
Preface	6
1 Introduction	9
1.1 Initial situation	9
1.2 Investigation concerns	9
1.3 Objective	9
1.4 Project reference	9
2 Technology	10
2.1 What is AI?	10
2.2 How does AI work?	11
2.2.1 Neurons	12
2.2.2 Activation function	12
2.2.3 Feedforward	13
2.2.4 Combining neurons into a neural network	14
2.2.5 Calculating loss	15
2.2.6 Back propagation	15
2.2.7 Example loss correction	17
2.2.8 Stochastic Gradient Descent	18
2.2.9 Visualization of neural nets	19
2.3 Large Language Models	20
2.3.1 Language Models	20
2.3.2 Recurrent Neural Networks	21
2.3.3 Transformers	21
2.3.4 En/Decoders	22
2.3.5 Summary	23
3 Development Chatbot	23
3.1 Main Features	24
3.2 Frontend	24
3.3 Backend	26
3.3.1 Embedding	28

3.4 Interaction cost calculation 28

3.5 Choosing the API 29

4 Summary 30

5 Index 31

1 Introduction

The client Learn4D365 is an online learning platform for the ERP-system Dynamics365. To help customers with different topics about their platform or the ERP-system, a support bot, that is capable of searching and using the knowledge the site offers in its full extend, is to be developed and made accessible on a user friendly fronted.

1.1 Initial situation

Learn4D365 offers no AI in their service, they have a web based learning platform without any integrated AI features. It features a navigable UI to find different courses around Dynamics365 and related topics. They offer a small popup-window for users to chat with a simple bot to help navigate, but this feature does not include any AI whatsoever. Since there are over 340 different courses available, with varying lengths and information, a bot that can use the available data to help users is a great example to show the capabilities of modern AI-systems.

1.2 Investigation concerns

To build a useful AI application for the learning platform, different technologies are to be studied and tested to build a optimized solution.

1.3 Objective

The objective of this thesis is to provide valuable insight into the different technologies one can use to achieve successful integration of AI into their services. Furthermore a proof-of-concept prototype for the learning platform is to be developed and tested.

1.4 Project reference

Rene Gayer is a client with whom I worked a few times in the past. He offered me the opportunity to work with him on this new technology to develop a solution for his business.

2 Technology

Since this thesis does not only involve the practical integration of AI into a solution for business, but also the research of different technologies, we face the following questions:

2.1 What is AI?

The term "artificial intelligence" is defined by the Cambridge-University as:

"The use or study of computer systems or machines that have some of the qualities that the human brain has, such as the ability to interpret and produce language in a way that seems human, recognize or create images, solve problems, and learn from data supplied to them." [11]

There is a very important distinction to be made when talking about AI. In general, for most people AI may be something that is human-like, meaning it "thinks" like a human. As it stands right now, this is, as far as we can say with our knowledge about the human brain and consciousness, very unlikely. The systems that are used today in a variety of places eg.: Autonomous driving, object recognition or for example the prominent "ChatGPT" are, as will be discussed later, essentially a product of mathematical theorems and huge amounts of processing power.

The AI technology we currently use is classified as weak-AI, or rather "narrow minded" AI. It does not think, understand nor reason like a human. It may act like it, as seen in in the Large language model (LLM) series from OpenAI, but it is not comparable to a human.

AI that has the capability to not just mimic human reasoning and understanding, would be called Artificial general intelligence (AGI).As it stands currently, there is no AI than can be classified as AGI in our world. Right now AGI is just a purely theoretical concept with no signs of anyone coming even close to creating true AGI. [7][3]

Further emphasizing the difference in AGI and our current AI is a thought experiment first published in 1980 by American philosopher John Searle.

"Searle imagines himself alone in a room following a computer program for responding to Chinese characters slipped under the door. Searle understands nothing of Chinese, and yet, by following the program for manipulating symbols and numerals just as a computer does, he sends appropriate strings of Chinese characters back out under the door, and this leads those outside to mistakenly suppose there is a Chinese speaker in the room." [3]



Figure 1: The chinese room argument [10]

Although there has been a lot of debate around Searles argument and its implications about strong AI, the main takeaway for this thesis is: the AI that is used in our current world does not "think" or "understand" as humans do, just like any algorithm or program it simply follows written instructions when given an input to produce an output.

Even though our current technology in AI may sound very primitive, it can achieve outstanding results when used properly. Various sectors in our society could benefit from proper integration of AI. In the following chapters of my thesis I will firstly expand on how exactly such AI systems work and then showcase various examples of practical applications AI is used in.

2.2 How does AI work?

This thesis is focused on a particular sub-field of artificial intelligence, in which the main goal is to simulate the topology of the human brain. This technique of trying to simulate the human layout of neurons is called Deep learning (DL). Deep learning is included in Machine learning (ML), which is a field in Artificial intelligence that describes training machines on data to create an algorithm, in stark contrast to classical programming, where algorithms are hard coded.[2]

2.2.1 Neurons

The basic premise of making an AI system with deep learning, is to firstly make an AI model or neural network. A neural network is in essence, modeled after the human brain, it has neurons that react to an input to give an output. A model can have a few neurons or massive amounts of neurons interlinked in layers with each other.

This is an explanation of how neural networks work mathematically. Creating such networks can be done in various technologies, a very popular way to make neural nets is with python. Python is a very beginner friendly programming language and has a huge repertoire of libraries to use. One of the the articles referenced in this thesis uses "numpy" as as library for the basic setup of a simple neural network.[17]

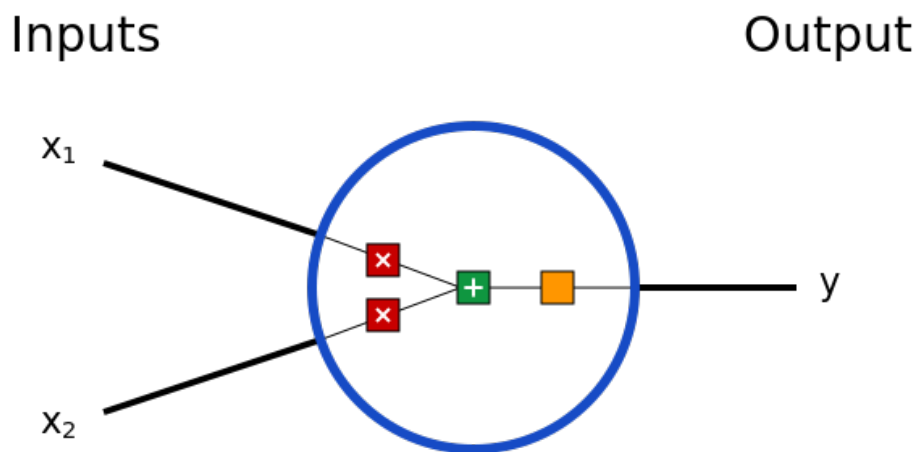


Figure 2: Basic neuron [17]

Figure 2 shows a neuron that takes two input values (x_1 , x_2) multiplies them with their respective weights (ω_1 , ω_2) and adds a bias (b).

$$\begin{aligned}
 x_1 &\rightarrow x_1 \cdot \omega_1 \\
 x_2 &\rightarrow x_2 \cdot \omega_2 \\
 &\rightarrow (x_1 \cdot \omega_1) + (x_2 \cdot \omega_2) + b
 \end{aligned}$$

2.2.2 Activation function

After arithmetic operations the result is passed into a activation function which is used to "compress" and turn the output into a predictable form. A commonly used function is the sigmoid function as seen on figure 3:

$$\rightarrow y = f((x_1 \cdot \omega_1) + (x_2 \cdot \omega_2) + b)$$

With the function $f(x)$:

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

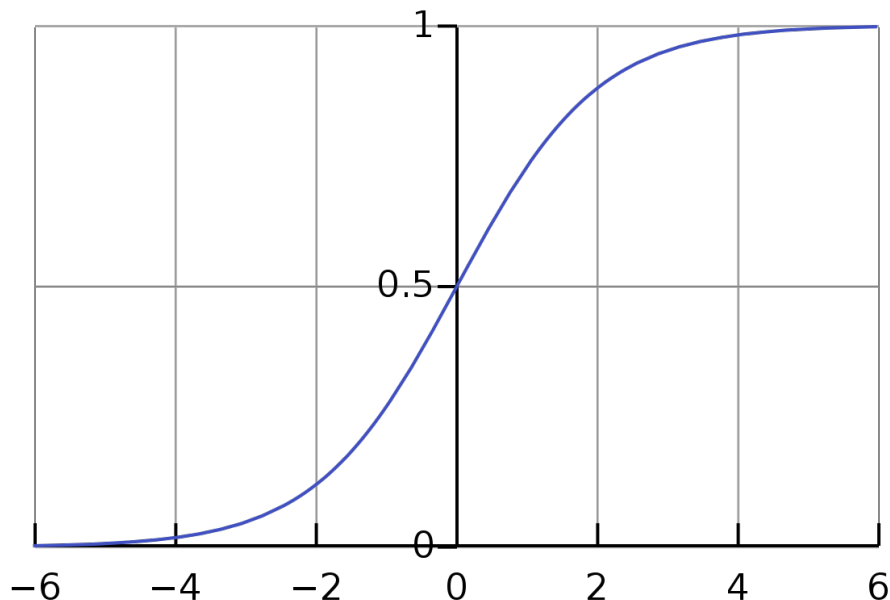


Figure 3: Logistic sigmoid function graph [15]

Essentially, the function compresses the output range of the neuron from $[-\infty; \infty]$ into the range of $[0; 1]$. Where large negative numbers become close to 0 and large positive numbers close to 1. In the following chapters the sigmoid function is used for example calculation, unless another function is explicitly stated.

2.2.3 Feedforward

Inputting a value into a neural net to get an output is called feedforward, for demonstration purposes I will give the previously defined neuron some arbitrary input. For this example I will define the parameters with:

The weights ω :

$$\omega = [0; 1]$$

The bias b :

$$b = 4$$

The inputs x as:

$$x = [2, 3]$$

This returns a result after arithmetic operations as:

$$(w \cdot x) + b = ((w_1 \cdot x_1) + (w_2 \cdot x_2)) + b = (0 \cdot 2) + (1 \cdot 3) + 4 = 7$$

Putting this into the activation function we get

$$y = f(w \cdot x + b) = f(7) = 0.999$$

2.2.4 Combining neurons into a neural network

A neural net can be seen as multiple neurons connected together:

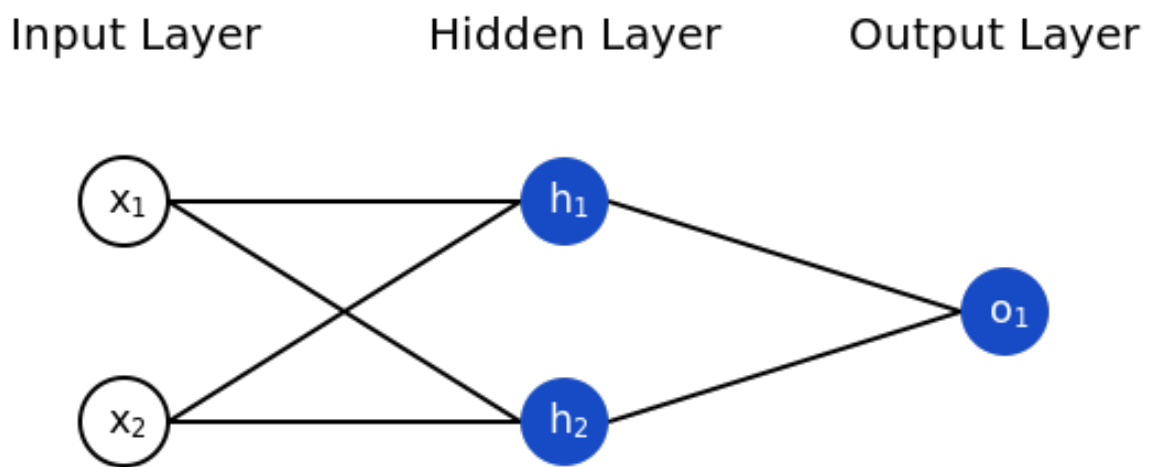


Figure 4: A simple neural network [17]

This particular net has 3 layers:

- An input layer that forwards the input values to the connected neurons.
- An hidden layer, which is inbetween input and output layer.
- An output layer that takes the resulting values of the hidden layer to give the final output.

A neural net can have any number of layers with any number of neurons within those layers. Calculating the output of this neural net, which is the output of the neuron o_1 , with the same parameters for every neuron:

The weights:

$$\omega = [0; 1]$$

The bias:

$$b = 0$$

With the input:

$$x = [2, 3]$$

Results in:

$$h_1 = h_2 = f(w \cdot x + b) = f((0 \cdot 2) + (1 \cdot 3) + 0) = f(3) = 0.9526$$

Since the parameters for every neuron are the same, the output of h_1 and h_2 is the same. Now we take those outputs and put them into the neuron o_1 as input:

$$o_1 = f(w \cdot [h_1; h_2] + b) = f((0 \cdot h_1) + (1 \cdot h_2) + 0)$$

$$\rightarrow o_1 = f(0.9526) = 0.7216$$

2.2.5 Calculating loss

In order to actually use the neural network for meaningful output, it needs to be trained first. In order to quantify if a network is getting better at the task it is being trained at, the error is calculated. For example the mean squared error (MSE) can be used:

$$\text{MSE} = \frac{1}{n} \sum_{i=1}^n (y_{\text{should}} - y_{\text{real}})^2$$

In essence this formula is used to get the average deviation of the real output to the output that the network is supposed to give.

- n is the number of samples
- y is used as the general variable for output
- y_{should} is the value the network is supposed to output when given an input
- y_{real} is the actual value the network puts out when given an input

The important part to remember is:

Training a network = trying to minimize its loss

2.2.6 Back propagation

After covering the basics of a neural network, how can neural networks be trained to actually give usable outputs when given an input? Weights and biases can be changed to alter the

results of the network, but how can they be changed to decrease the loss or rather, how can they be influenced to make y_{real} as close as possible to y_{should} when given a sample? I am going to use the same neural net as in the previous example loss calculation (Figure 4). Firstly, the loss can be defined as a multivariable function:

$$L(\omega_1, \omega_2, \omega_3, \omega_4, \omega_5, \omega_6, b_1, b_2, b_3)$$

For example, how would ω_2 influence loss if it were changed? With a partial derivative this change can be predicted:

$$\frac{\partial L}{\partial \omega_2}$$

Using the chain rule to add another partial derivative into the equation results in:

$$\frac{\partial L}{\partial \omega_2} = \frac{\partial L}{\partial y_{real}} \cdot \frac{\partial y_{real}}{\partial \omega_2}$$

Since $L()$ is known:

$$\rightarrow \frac{\partial L}{\partial y_{real}} = \frac{\partial (y_{should} - y_{real})^2}{\partial y_{real}} = -2(y_{should} - y_{real})$$

Now only the second partial derivative is left, ω_2 only affects h_1 so:

$$\frac{\partial y_{real}}{\partial \omega_2} = \frac{\partial y_{real}}{\partial h_1} \cdot \frac{\partial h_1}{\partial \omega_2}$$

$$\rightarrow \frac{\partial y_{real}}{\partial h_1} = \omega_5 \cdot f'(\omega_5 h_1 + \omega_6 h_2 + b_3)$$

The last term:

$$h_1 = f(\omega_1 x_1 + \omega_2 x_2 + b_1)$$

$$\rightarrow \frac{\partial h_1}{\partial \omega_2} = x_2 \cdot f'(\omega_1 x_1 + \omega_2 x_2 + b_1)$$

Deriving the sigmoid function:

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

$$f'(x) = \frac{e^{-x}}{1 + e^{-x^2}} = f(x) \cdot (1 - f(x))$$

Putting all this together, the partial derivative can be expressed as:

$$\frac{\partial L}{\partial \omega_2} = \frac{\partial L}{\partial y_{real}} \cdot \frac{\partial y_{real}}{\partial h_1} \cdot \frac{\partial h_1}{\partial \omega_2}$$

This system of calculating partial derivatives by working backwards is known as backpropagation.

2.2.7 Example loss correction

So y_{should} is the desired value a neural net **should** output when given an input. For example, take a neural net and when given an input $x_{1,2} = [3, 2]$ the neural net with the randomly chosen starting weights of $\omega = [1, 1]$ and a bias $b = 1$, which are the same for every node, **should** output the arbitrary chosen value of $y_{should} = 0$.

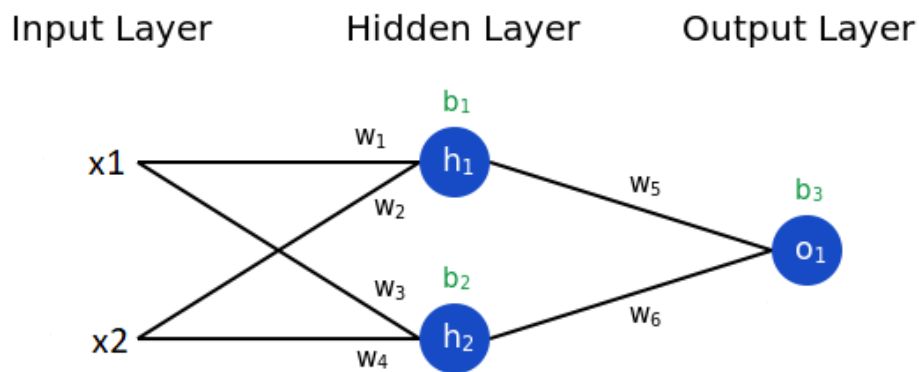


Figure 5: Neural net with weights and biases [17]

The parameters shown in figure 5 were defined as:

$$\begin{aligned} x_1 &= 3 \\ x_2 &= 2 \\ \omega_1 &= \omega_3 = \omega_5 = 1 \\ \omega_2 &= \omega_4 = \omega_6 = 1 \\ b_1 &= b_2 = b_3 = 1 \end{aligned}$$

Output of h_1 and h_2 :

$$h_1 = h_2 = f((1 \cdot 3) + (1 \cdot 2) + 1) = 0.997527$$

Output of the whole net:

$$y_{real} = o_1 = f((\omega_5 \cdot h_1) + (\omega_6 \cdot h_2) + b_3) = f((1 \cdot 0.997527) + (1 \cdot 0.997527) + 1) = 0.95235$$

The real output is not quite the output desired. The MSE is:

$$MSE = \frac{1}{1} \sum_{i=1}^1 (0 - 0.95235)^2 = 0.906971$$

Calculating the partial derivative to adjust ω_2 to decrease loss:

$$\frac{\partial L}{\partial \omega_2} = \frac{\partial L}{\partial y_{real}} \cdot \frac{\partial y_{real}}{\partial h_1} \cdot \frac{\partial h_1}{\partial \omega_2}$$

$$\frac{\partial L}{\partial y_{real}} = \frac{\partial(0 - y_{real})^2}{\partial y_{real}} = -2(0 - 0.95235) = 1.9047$$

$$\frac{\partial y_{real}}{\partial h_1} = \omega_5 \cdot f'(\omega_5 h_1 + \omega_6 h_2 + b_3) = 1 \cdot f'(1 \cdot 0.997527 + 1 \cdot 0.997527 + 1) = 0.045379$$

$$\frac{\partial h_1}{\partial \omega_2} = x_2 \cdot f'(\omega_1 x_1 + \omega_2 x_2 + b_1) = 2 \cdot f'(1 \cdot 3 + 1 \cdot 2 + 1) = 0.004933$$

$$\frac{\partial L}{\partial \omega_2} = 1.9047 \cdot 0.045379 \cdot 0.004933 = 0.000426$$

Since the ratio is positive, decreasing ω_2 will also decrease loss. Since:

$$\partial L = 0.000426 \cdot \partial \omega_2$$

for this current situation of the neural net.

Because the factor is very small, it will decrease the loss only by a small amount. Of course, it is possible to make $\partial \omega_2$ bigger, but this could cause the exact opposite. By incrementing ω_2 in big steps, it is possible to overshoot the target.

2.2.8 Stochastic Gradient Descent

Stochastic Gradient Descent (SGD) is a method of optimizing the variables of a neural net. In the most basic terms, SGD does the same procedure of changing a weight or bias as described in previous subsection. Instead of only adapting one weight or bias, it optimizes every parameter for the given sample in a defined step. In essence, it is this update function:

$$\omega_2 \leftarrow \omega_2 - \eta \frac{\partial L}{\partial \omega_2}$$

η is a constant called "learning rate" that controls how "fast" the neural net is trained, or rather, it decides how much the individual parameters are adjusted per cycle of adapting the parameters. If the ratio of Loss to weight is positive, ω_2 will decrease, which will decrease Loss. On the opposite, if the ratio of Loss to the weight is negative, ω_2 will increase, which will decrease Loss.

It is important to choose an appropriate learning rate since incrementing or decrementing

the individual parameters too far per learning cycle can cause the neural net to overshoot the optimal point for its practical use.

2.2.9 Visualization of neural nets

A good resource of visualization about how a neural network learns is freely available online in a tool created by Daniel Smilkov and Shan Carter: Tensorflow neural net playground[9]. To demonstrate the importance of an adequate learning rate, I will compare two identical neural nets:

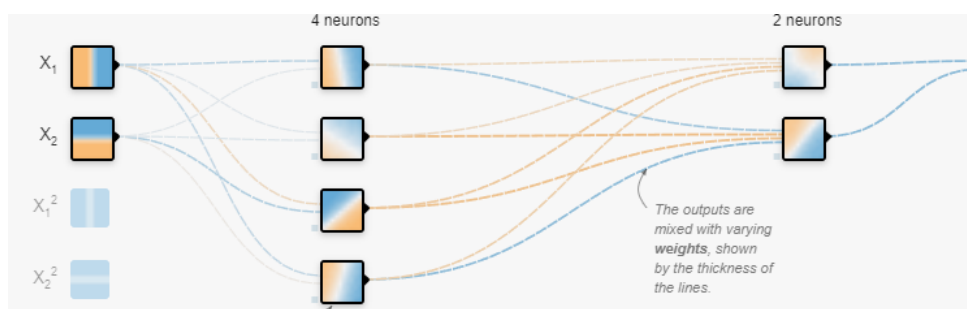


Figure 6: Neural net used for comparison of learning rates[9]

Adjusting the learning rate shows how "overshooting" of the target can occur:

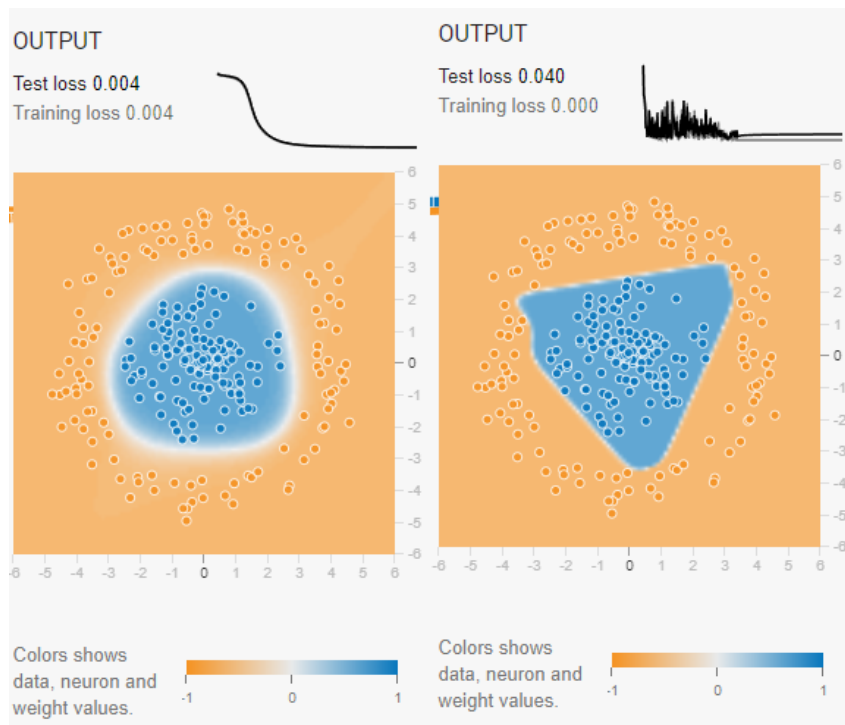


Figure 7: Output of two neural nets trained with different learning rates[9]

Both neural nets were trained to classify figures in a sample, in this case the blue and orange dots shown in figure 7. The left net was trained with $\eta = 0.01$ and the right net

with $\eta = 0.3$. When analyzing the above shown figure, there is a clear difference in the output classification. The left network creates a clear round perimeter around the blue dots whereas the right net does not create such a clear distinction between the orange and blue dots, some of the ends of the created perimeter come very close to the orange dots. The graph above the visualized output shows a very "clean" drop of loss for the left net with no swinging between the training cycles. In contrast, the right net shows very large fluctuations in loss with each iteration and then settles at a constant higher loss than the left net. Both nets were trained for about 350 cycles.

2.3 Large Language Models

As previously mentioned, the subsection of neural nets called LLMs have gotten very prominent in recent years, especially since the release of the service "ChatGPT" by OpenAI. Just as the name suggests, LLMs are capable to "understand" human language and to output in context to the input, in this case normal text.

2.3.1 Language Models

LLMs are a subsection of language models. Language models aim to predict and generate plausible language, for example "auto complete" is a language model. Different words or characters are numbered and classified as tokens (this method of classifying words or characters in tokens is not used in every language model, but for the sake of this explanation it suffices to use it). A model will try to predict what token to insert into a sequence of input tokens[4]. For example:

When I hear rain on my roof, I . . . in my kitchen.

If you assume that a token is a word, then a language model determines the probabilities of different words or sequences of words to replace the 3 dots. For example, a language model might determine the following probabilities:

Cook soup 9.4%
Warm up a kettle 5.2%
Cover 3.6%
Nap 2.5%
Relax 2.2%

Instead of inserting a single token, a model could also predict whole sentences, paragraphs or even larger sequences of tokens.

Since the human language is a very complex and large structure, modeling the whole of it is quite resource-intensive. Modern services that use LLMs not only predict one word, but whole paragraphs worth of text, to facilitate such a large output huge neural networks are used. BERT and PaLM 2, both models introduced by Google, use 110 Million and 340 Billion parameters respectively. OpenAI's GPT4 has, supposedly, about 1.8 Trillion parameters [8], this number has not been officially confirmed but it is believed the Model uses multiple Models combined with each other for different cases (Mixture of experts (MoE)). Each one of these models have been described as LLMs, so the definition of large in large language model is very vague. Large not only references the parameters of the model but also the datasets used to train such networks.

2.3.2 Recurrent Neural Networks

The first predominately used architecture for LLMs was Recurrent Neural Network (RNN). RNNs process information in a sequential manner, one word at a time, capturing dependencies and context. For example:

I love ice cream.

A RNN starts by processing the word "I" and then continues down the sequence, one word at a time. When reaching the next word, in this case "love", it takes the context and the relation of the word "I" into account. The network retains a hidden state in which the information of the previous words is stored, this allows it to consider the meaning of the whole text. The hidden state is updated with each step and makes these networks especially suitable for machine translation and speech recognition. [1]

2.3.3 Transformers

In 2017, a new architecture was introduced called Transformers, it was designed around a key feature: Attention. The new architecture made it possible to focus on relevant information in given inputs. Transformers use a technique called "self-attention" to give the relation of each token to another token tangible worth [4] [12] [1]. In the official paper by google "Attention Is All You Need" the architecture is introduced with:

"The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show

these models to be superior in quality while being more parallelizable and requiring significantly less time to train.” [12]

To display the difference between RNNs and Transformers, take the following example:

The animal didn't cross the street because it was too tired.

There are 11 words in the preceding sentence, so each of the 11 words is paying attention to the other ten, wondering how much each of those ten words matters to them. For example, notice that the sentence contains the pronoun **it**. Pronouns are often ambiguous. The pronoun **it** always refers to a recent noun, but in the example sentence, which recent noun does it refer to: the animal or the street? The self-attention mechanism determines the relevance of each nearby word to the pronoun **it**.

Transformers do not rely on sequential processing, they can do computation in parallel and process sequences of tokens more efficiently. Each of the words in a given sentence or text is broken down into tokens and each token is represented as a vector that displays the tokens meaning and context. Transformers are trained to analyse the context and relationships between tokens to "understand" the overall meaning.

LLMs are specific types of models that are built upon the transformers architecture, transformers can be used for different tasks other than language modeling. LLMs are focused on text generation / language modeling and transformers are the underlying architecture that allows LLMs to "understand" text, by capturing the contextual relationships and long range dependencies of the input [1].

Every modern LLM depends on the transformer architecture: the GPT series, all models from Google and other competitors. The introduction of the "self-attention" method fabricated a huge leap in language models and made it possible to build the truly "usable" language models we know today.

2.3.4 En/Decoders

Transformer models can also be divided into three categories: encoders, decoders, and encoder-decoder architectures. This categorization is based on the different roles these components play in the model's overall function. Encoders aim to understand the input sequence. They focus on processing the input and capturing its meaning and context. Decoders, on the other hand, generate output based on the information learned by the encoder. They take the encoded representations and produce the desired output sequence. Encoder-decoder models combine both encoder and decoder components. They are used in tasks where the input and output sequences have different lengths or meanings. The

encoder understands the input sequence, and the decoder generates the corresponding output sequence. [1]

Type	Example Models	Example use cases
Encoder Only Models	BERT ROBERTA	Sentiment Analysis Named Entity Recognition Word Classification
Encoder Decoder Models	T5 BART	Translation Text Generation Question Answering
Decoder Only Models	GPT Bloom	Text Generation

Figure 8: Different models with Encoders and or Decoders[1]

2.3.5 Summary

The important part to take away from the previous sections is that AI models are essentially just universal function approximators. Models are trained on a set of input and "told" what output they should produce. The input and output can be anything: text or pixels and so on. In the end the model only gets zeros and ones as an input and outputs zeros and ones. Now taking this concept and pairing it with huge amounts of datasets and computing power paints the picture of modern AI. By sheer brute strength of available infrastructure, it is possible to even build models that are able to simulate the ability of comprehension. But again, there is no understanding or reasoning as we do, it is just an emulation of the human mind at best. But still, by using this technology a lot of useful applications can be build. As previously mentioned, I will now explain a practical implementation of a LLM:

3 Development Chatbot

Since this prototype is not optimized and only a proof of concept to show possible applications of modern AI technology, room for improvement is a given. The client's intention was on only including the basics for the first versions to get a usable bot with the simplest means possible. That is why a simple HTTP-Server with PHP was chosen instead of other more sophisticated technologies. The version documented in this thesis can be used in a production environment, but as previously mentioned, is only one of the first usable iterations. I will continue to work with the client on this solution, but for the extend of this thesis I will focus on the existing prototype.

3.1 Main Features

The main features of the supportbot were defined as follows:

- Able to provide a customized interaction with the customer / converse with the customer
- Search functionality to find specific resources the end-user may find interesting
- Use found data as a source to help the customer with his or her request
- Custom frontend for user interaction
- Able to run on provided HTTP-server with basic PHP support

The general scheme of how the bot functions is as depicted below:

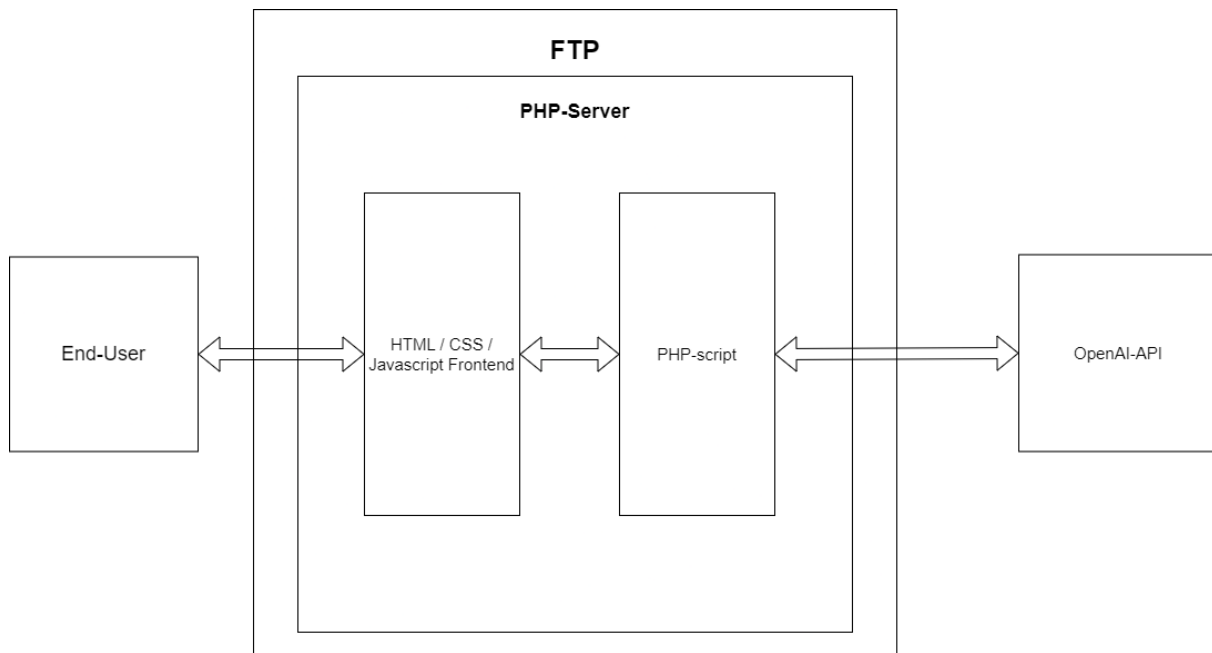


Figure 9: General function schema of frontend and backend

3.2 Frontend

A simple HTML page with CSS styling and JavaScript to handle the chat interface is used for the UI.

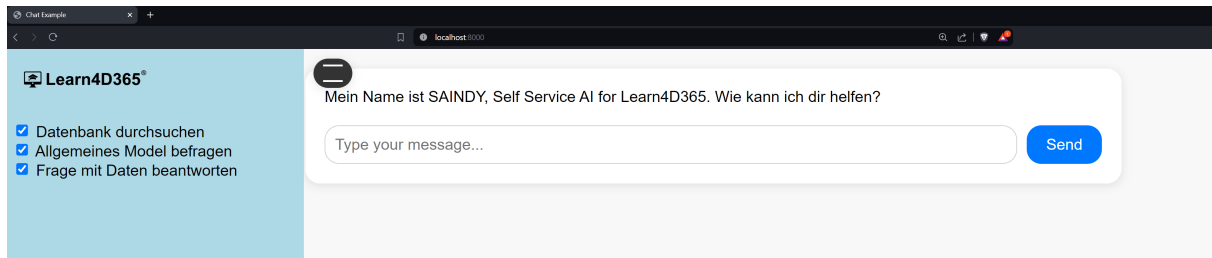


Figure 10: Locally used frontend interface

The logic of the frontend was implemented in JavaScript, the user can interact with the chatbot via a message-field. The frontend checks for an empty message and ongoing requests that have not finished before sending the message. When sending a message, the frontend sends the raw text with information about the checked boxes in a JSON formatted string included in a HTTP request to the server. The key method of how the communication between the UI and the backend was achieved was per XMLHttpRequest:

```
1 //A XMLHttpRequest object is declared:
2 const xhrPHP_Master = new XMLHttpRequest();
3
4 //After several conditions are met, a POST request is sent:
5 xhrPHP_Master.open("POST", "API_handler.php", true);
6 xhrPHP_Master.setRequestHeader("Content-Type", "application/json");
7 xhrPHP_Master.setRequestHeader("Search_DB_enabled", checkbox1.checked);
8 xhrPHP_Master.setRequestHeader("Request_GModel", checkbox2.checked);
9
10 // Send the request with the input data
11 xhrPHP_Master.send(JSON.stringify({ "role": "user", "content":
    userInput.value, "Search_DB_enabled": checkbox1.checked,
    "Request_GModel": checkbox2.checked, "Solve_Question": checkbox3.checked
  }
});
```

The status of two check-boxes is included in the header, because originally the states of the boxes were supposed to be sent in the header, but testing on the live server showed, that the HTTP-provider did not forward custom headers to the script. Thus, the states are sent in the actual JSON payload. Other than handling the communication between user and backend, the frontend does no particular important job for the function of the bot. Most of the logic for the features, like searching the DB for relevant resources, is in the PHP file. When a message is sent, the interface awaits the backend's response, multiple HTTP requests are made in the background till the full response of the bot is finalized and then displayed in the chatbox. In essence, the JavaScript just awaits the backends responses and sends another HTTP request if the response is not final. The amount of HTTP requests

can vary depending on the number of checked features eg: "Search DB" or "Answer question with found data". To make the interaction look smoothly, the responses of the bot are displayed character per character with a small delay to make the bot seem more responsive.

3.3 Backend

Since the final product would be hosted on a HTTP-Server with PHP support, the bot was tested on a local PHP server instance. With:

```
1 php -S localhost:8000
```

a local PHP server instance can be started to test PHP code. It is required to install a PHP distribution locally to run a test server. I used the package included in XAMPP [6].

The backend handles all of the primary logic:

- Embedd the user query with an OpenAI embedding model to compare the returned vector with the stored vectors
- Saving previous messages of the conversation to be able to send the whole conversation-history to the model
- Handles the state of the answer: is the request / answer finished or does the backend need to search for data
- Sent found data to model with user question, if user requested help

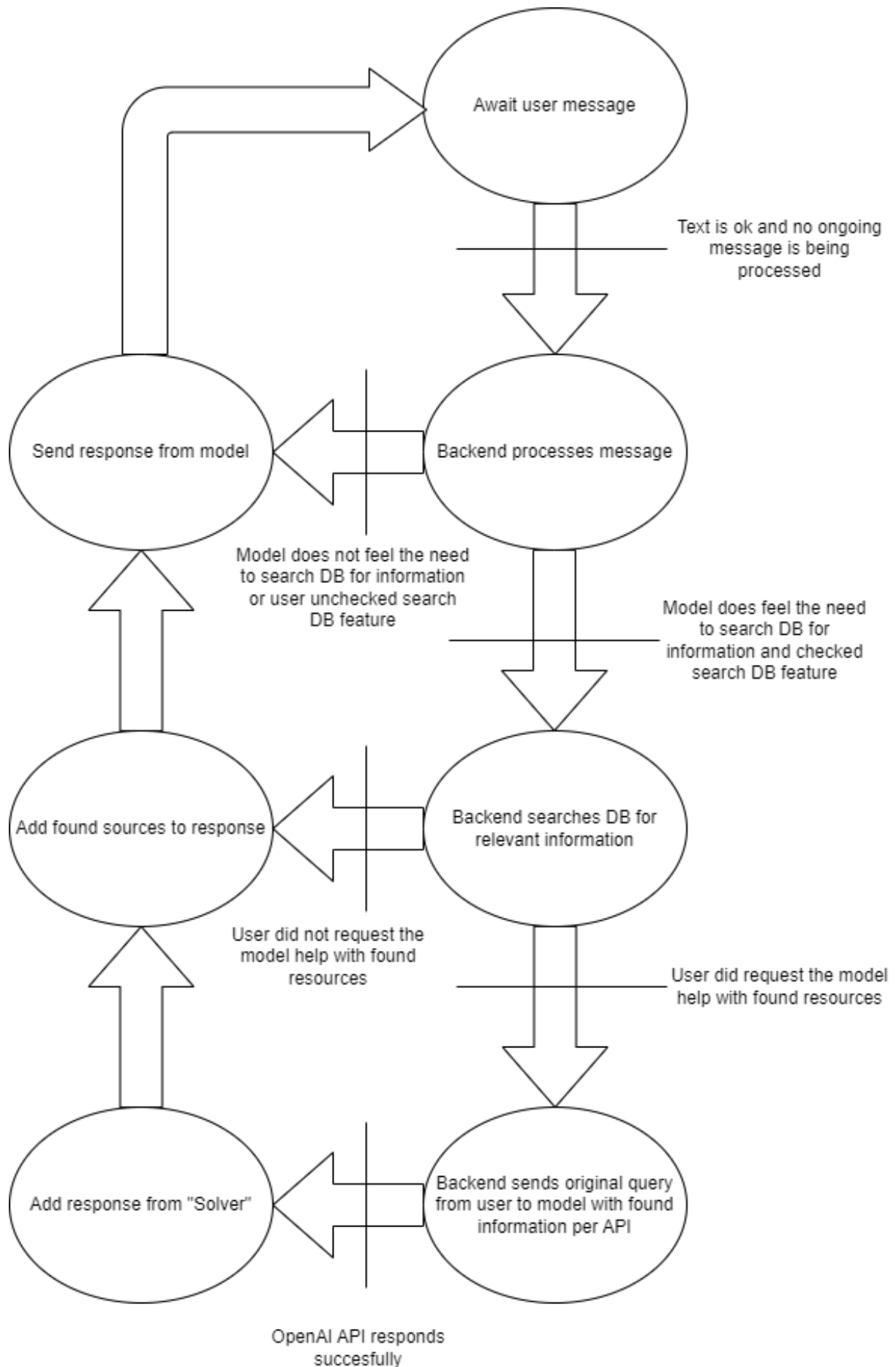


Figure 11: Backend scheme

3.3.1 Embedding

In order to find relevant data for a given user query, the text is embedded and compared with a list of stored data. In essence: a text can be embedded with a neural net to output a vector with n dimensions. Per cosine similarity, two vectors can be compared and their similarity can be calculated. The higher the cosine similarity, the more likely it is that two given texts / vectors are related to one another:

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\| \|\mathbf{B}\|} = \frac{\sum_{i=1}^n A_i \times B_i}{\sqrt{\sum_{i=1}^n A_i^2} \times \sqrt{\sum_{i=1}^n B_i^2}}$$

When the LLM decides that the backend should search for relevant DB entries, the backend sends the question / query of the user to be embedded by the embedding models from OpenAI. The vector that the embedding models returns from the API, is compared to every other stored vector in the Database. After comparing each entry, the backend sorts for the five highest similarity percentages and returns a list to the frontend. The frontend then searches for the entries in a JSON file and copies the id of the course and the name into the chat-box. The backend also adds the found entries into a "conversation-history" array so the found results can be used for context in the next query. Currently the used "Database" is a JSON document, since that was the easiest way to implement a simple DB. In the future a fully fledged vector database is planned, with a user tool to add / update new entries.

If the user requested the model help with the found data, the original query of the user with the full content of the found sources is sent to a LLM per API. The answer is then copied into the chatbox and also into the "conversation-history" array, so the context can be used for further queries.

3.4 Interaction cost calculation

Since the goal of this prototype is to also show viability in a actual live production environment, the costs of running the bot had to be estimated. Currently the gpt-4-0125-preview model of OpenAI is used for the bot. The cost of an API call is, as it stands at the time of this thesis, 10.00\$ per 1 million token of input and 30.00\$ per 1 million token of output.

The Prompt used as the system prompt, that is sent every time the user interacts with the bot, is 645 token (The amount of token can be checked with the a free online tool: Tiktokenizer[5]). Depending on the user query the token sent in the first interaction may average around 700. This would infer a cost of about 0.007\$ or about 0.7 cents. Now depending on if the model checks the DB with an embedding, the cost for a embedding is 0.10\$ per 1 million token. Assuming that a normal question from a user is about 10 token, the cost comes to around 1e-7\$, which is neglectable. If the bot uses the context stored

in the DB to answer the question, the costs can vary widely since the stored courses are not uniform in length. From testing, a general token amount of about 1000 to 2000 token is observed, but again, depending on the question vastly different amounts could be used. So the cost of input for one full interaction with context search and answer is around 0.027\$. The output token of the bot can also vary widely, though they can be limited in the API call, but generally they are around 20 token for the first response and around 100 to 500 token for the context answer, so the cost of the output is around 0.0156\$ for the whole first interaction. Adding the input cost and the output cost together, the sum of costs is around 0.0426\$. Now again, I want to emphasize that this can vary widely depending on user query, also this example calculation is assuming the worst case. In practice, while testing, the costs were about $< 0.01\$$ for one initial interaction.

Since the whole conversation with all of the data needs to be saved and sent to the bot again when another query is made in the same chat, the expenses add up additively. For example if the first interaction cost 0.01\$ then the next interaction would cost 0.01\$ + the token cost of the next query (Actually the old output is also used as input and thus the price of the previous output is reduced to 1/3, so the cost is not directly additively but it suffices as a general rule). After a few user queries this cost can rise up quite significantly and if a few thousands users are using the bot, the costs can increase very fast. But the expenses of using these models are decreasing quite rapidly and since the start of development, the costs have decreased to about a quarter of the initial costs in just 1 year. So in the future, the expenses will more than likely follow this decreasing path.

3.5 Choosing the API

OpenAI was chosen, because at the time their models were the only ones readily available per API. Also their models were leaps and bounds ahead of competition like Google. The release of GPT-4 made the bot possible, since no other model at the time was able to handle user queries with system instructions as well as GPT-4. Their embedding models were the most effective and the client had already gotten API access from OpenAI for GPT-4. Waiting for access to another API eg.: Google would have delayed development for an unknown time. The competition is just now catching up with new models, but still, OpenAI capitalized on its huge success upon the release of ChatGPT and to this day is at the forefront of AI applications. As it stands now, there are no plans to move away from OpenAI.

4 Summary

As hopefully shown in this thesis, the world of AI is undergoing rapid change. AI has been used in a lot of sectors of our daily lives to date and modern services like ChatGPT will continue to be incorporated into our world. Especially LLMs will probably predominantly be used for a variety of applications, since they can in essence simulate "intelligence" and "reason" as a human would. The important part here is, that right now, it is just that: simulation. These models are not alive, nor are they thinking, it is the combination of mathematics, randomness and computing power that makes it seem like these models are conscious.

As for the practical trial of using modern AI in a solution: In the end, a functional and usable bot that can search/use custom data supplied by the client was created. The bot is hosted as a Beta version online for testing and showcase purposes and will be developed further in the future. All the features the client requested were built and shipped with the prototype, also different technologies were researched and different "AI-Providers" were compared and the optimal provider was chosen for the prototype.

5 Index

Figures

1	The chinese room argument [10]	11
2	Basic neuron [17]	12
3	Logistic sigmoid function graph [15]	13
4	A simple neural network [17]	14
5	Neural net with weights and biases [17]	17
6	Neural net used for comparison of learning rates[9]	19
7	Output of two neural nets trained with different learning rates[9]	19
8	Different models with Encoders and or Decoders[1]	23
9	General function schema of frontend and backend	24
10	Locally used frontend interface	25
11	Backend scheme	27

Glossary

AGI AGI, or general AI, is a theoretical form of AI where a machine would have an intelligence equal to humans; it would be self-aware with a consciousness that would have the ability to solve problems, learn and plan for the future. [7].

DL Deep learning is a subfield of Artificial intelligence included in Machine learning in which a virtual topology comparable to that of the brain is simulated. [2].

LLM A large language model (LLM) is a language model notable for its ability to achieve general-purpose language generation and understanding [13].

ML ML is a field that focuses on the learning aspect of AI by developing algorithms that best represent a set of data. In contrast to classical programming, in which an algorithm can be explicitly coded using known features, ML uses subsets of data to generate an algorithm. [2].

MoE Mixture of experts (MoE) is a machine learning technique where multiple expert networks (learners) are used to divide a problem space into homogeneous regions. It differs from ensemble techniques in that for MoE, typically only one or a few expert models are run for each input, whereas in ensemble techniques, all models are run on every input. [14].

RNN Type of neural network used for sequential data processing, including natural language tasks. RNNs process information in a sequential manner, one word at a time, capturing dependencies and context. [1].

SGD Stochastic gradient descent (often abbreviated SGD) is an iterative method for optimizing an objective function with suitable smoothness properties (e.g. differentiable or subdifferentiable). [16].

References

- [1] Vijay Chaudhary. *Transformers and LLMs: The Next Frontier in AI*. June 2023. URL: <https://www.linkedin.com/pulse/transformers-llms-next-frontier-ai-vijay-chaudhary>.
- [2] Rene Y. Choi et al. "Introduction to Machine Learning, Neural Networks, and Deep Learning". In: *Translational Vision Science Technology* 9.2 (Feb. 2020), pp. 14–14. ISSN: 2164-2591. DOI: 10.1167/tvst.9.2.14. eprint: https://arvojournals.org/arvo/content_public/journal/tvst/938366/i2164-2591-226-2-2007.pdf. URL: <https://doi.org/10.1167/tvst.9.2.14>.
- [3] David Cole. "The Chinese Room Argument". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward N. Zalta and Uri Nodelman. Summer 2023. Metaphysics Research Lab, Stanford University, 2023.
- [4] Google for Developers. Aug. 2023. URL: <https://developers.google.com/>.
- [5] dqbd. *Tiktokenizer*. 2024. URL: <https://tiktokenizer.vercel.app/>.
- [6] Apache Friends. *XAMPP*. 2024. URL: <https://www.apachefriends.org/de/index.html>.
- [7] IBM. 2024. URL: <https://www.ibm.com/topics/artificial-intelligence>.
- [8] katerinaptrv. *GPT4- All Details Leaked*. June 2023. URL: <https://medium.com/@daniellefranca96/gpt4-all-details-leaked-48fa20f9a4a>.
- [9] Daniel Smilkov and Shan Carter. *Tensorflow neural net playground*. 2024. URL: <https://playground.tensorflow.org/>.
- [10] Alan Tan. *A Chinese Speaker's take on the Chinese Room*. 2019. URL: <https://towardsdatascience.com/a-chinese-speakers-take-on-the-chinese-room-88a0558b2cc8>.
- [11] Cambridge University. 2024. URL: <https://dictionary.cambridge.org/de/worterbuch/englisch/ai>.
- [12] Ashish Vaswani et al. "Attention is All You Need". In: 2017. URL: <https://arxiv.org/pdf/1706.03762.pdf>.
- [13] Wikipedia contributors. *Large language model — Wikipedia, The Free Encyclopedia*. [Online; accessed 27-February-2024]. 2024. URL: https://en.wikipedia.org/w/index.php?title=Large_language_model&oldid=1210546218.
- [14] Wikipedia contributors. *Mixture of experts — Wikipedia, The Free Encyclopedia*. [Online; accessed 12-March-2024]. 2024. URL: https://en.wikipedia.org/w/index.php?title=Mixture_of_experts&oldid=1211969795.

- [15] Wikipedia contributors. *Sigmoid function* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 27-February-2024]. 2024. URL: https://en.wikipedia.org/w/index.php?title=Sigmoid_function&oldid=1196609098.
- [16] Wikipedia contributors. *Stochastic gradient descent* — *Wikipedia, The Free Encyclopedia*. [Online; accessed 12-March-2024]. 2024. URL: https://en.wikipedia.org/w/index.php?title=Stochastic_gradient_descent&oldid=1213033008.
- [17] Victor Zhou. *Machine Learning for Beginners: An Introduction to Neural Networks*. 2022. URL: <https://victorzhou.com/blog/intro-to-neural-networks/>.