Mohammed VI Polytechnic University

Institute of Science Technology & Innovation

ALKHWARIZMI Department

Project Report

Question Generation, AND Long Form Open Domain Question Answering

Conducted BY:

- **Anouar TAIZOUKT**

Supervised By:

- **Pr. Fahd KALLOUBI**

Academic Year: 2022/2023

Abstract

Reading a book, an article or any kind of documents was and still one of the tasks that students, teachers, researcher have to do at least once a day in order to prepare for an exam, to complete an assignment, or to understand something or sometimes just for discovering or learning something new. Which is a very challenging and exhausting exercise to perform daily. Especially when the reader is looking for precise information in a big document, in this case he will have to read the whole, or a big part of the document in order to extract what he needs, and of course we can't neglect the fact sometimes while reading a book we might not find totality of what we are looking for, which means the reader might have to read more than one document in order to find what he is looking for exactly, which is both very tyring and time wasting.

One of the solutions that we might think at firstly is to offer some summarized versions of documents do people won't have to go through the long process of reading the whole document, or even to build some online communities in which we can ask people about information from a document they have already read but despite the progress offered by this two solutions it would be better and faster if we can solve this problem without counting on someone else's help.

With the appearance of the Artificial intelligence (AI) and especially it's sub-domains Machine Learning and Deep Learning we can think of another solution for to make the process of extracting information from documents really fast and efficient specially when we consider the performance and the results of the Deep Learning based methods.

Considering the (DL) based methods, the documents can be seen as an input to models.

Those models are supposed to be trained on performing the task of extracting information and generate answers form the given document based on a question provided by the user, and even if the reader can't think of a question that represents what he really needs from the document, we can build a model that can automatically generate a list of questions to make the process even faster.

Key Words: Deep Learning, Generating Answers, Generating questions.

Acknowledgement

After finishing the work on this project, I am glad to express my very big gratitude to everyone who helped me while developing this work, especially Pr. Fahd KALLOUBI my supervisor, and my professor, who I want to thank firstly for suggesting a such important and very motivating topic to work on, and also for his availability and his advices that were very crucial for me to understand and to take the right steps while working on each one of this project’s dimensions (Question answering and Question generation).

And special thanks to Mr. MOKERE Reda from the ESAFE department for the time he dedicated to try this application and for providing a very helpful feedback.

Table Of contents

Table of figures

List Of Abbreviations

Abbreviation	Original
RNN	Recurrent Neural Network
NLP	Natural Language Processing
LSTM	Long short-term Memory
DL	Deep Learning
ANN	Artificial Neural Network
Sec2sec	Sequence to sequence
Bert	Bidirectional Encoder Representations from Transformers
T5	Text-to-text Transfer transformer
MaxEnt	Maximum Entropy
GPT	Generative pre-trained transformer
BPTT	Backpropagation through time
RTRL	Real time Recurrent learning
GRU	Gated recurrent unit

Introduction

Reading is definitely one of the best ways to develop a creative mind, and it's also one of the greatest activities that someone consider doing nowadays considering all the other fruitless habits that the most of us are adopting lately.

But sometimes although all its advantages, reading might not be the best way to learn, understand or to get an information from a document, for example it might take a very long time for a student to retrieve an exact information from an article because he might have to understand all of it in order to get the full information he wants, or for person who already read a document but forgot a part of it, he might have to reread a considerable part of the document to find what he is looking for exactly, and we can much more examples where reading a document in order to extract an information might be very tyring and also time consuming.

To solve this kind of problems and in order to allocate more time to using the information instead of looking for it, researchers in the field of Artificial intelligence thought on way in which we can use the ability of Deep Learning tools to make the machine able to understand a text input the same way human can, by understanding the relation and the dependencies between the words of text, and they were and are having some very good results starting with the RNNs and after the LSTMs and finishing with the very powerful and the extremely well performing in the field of NLP, the transformers.

This report describes a web application that uses tools that are manly based on transformers, to provide a solution for the problem that we discussed by giving its users the ability to get answers to their open domain questions by providing a context, and they can also automatically generate questions based on the provided context.

The main objective of this report is to provide the necessary explanations to all the tools used to build the models that we can use to achieve our goal which is being able to answer and generate questions based on a given context, to provide an easy-to-understand workflow of the application and also to present a better strategy to generate a long form answers.

To summarize we can determine the objective of this work on three points:

- Provide a well explained answers to a user's questions.
- Generate questions from a given document.
- Keeping the documents provided by users and use it to provide better and rich answers to future users.

I- Background

In this section we will be explaining important notions that needs to be understood by providing a deep clarification in order to guarantee a better following in the next sections.

1.1 Question answering

1.1.1 Question answering types.

We can distinguish between two types of Question answering, which are the normal question answering and the long form question answering, and the main difference between these two types is that the first one is generally used to answer direct simple questions that requires simple answers, like (yes/no) questions or questions like « **what's the nearest restaurant?** » and it retrieves the answers from a given text.

The second type of question answering is the long form question answering, and it aims to not only provide simple answers to the asked questions, but also to give some explanation to the given answers, and to perform this task, LFQA models search for a large external knowledge from a provided text or on the web to generate a paragraph length and well explained answer.

1.1.2 Question answering methods.

To perform the question answering tasks we can consider two types of DL models:

- **Extractive Models**

  Extractive models are based on extracting relevant information from a given document to answer the asked questions, and to provide more information about the answer, this type of models extracts the information multiple documents.

- **Abstractive Models**

  Abstractive models are a combination of sequence-to-sequence models and language models, and it aims to retrieve information about the asked question from a given text (documents, html pages …) and then generate a paragraph length answer based on the knowledge they got by synthesizing the information source.

  In this case sequence-to-sequence are used to synthesize the text the give information sources (documents, web pages), while the language models are trained to predict the questions, web pages where we can find the answers and the answer.

1.2 Question generation

Automatic question Generation is one of the NLP approaches that aims to generate several types of questions based on a given Corpus, and it can be very helpful specially for education by helping professors on generate quizzes automatically, moreover students can also use question generation techniques for self-analysis.

Furthermore, Automatic question Generation is an area of research where many researchers have presented their work and is still an area under research to achieve higher accuracy.

There are several approaches that can be used for question generation and automatic question generation:

- **Answer aware question generation.**

  In answer aware question generation, our model is asked to generate questions from a given context, but before starting this process it needs also a given answer.

  In order to extract answers from a document in order to start the question generation process we can use several methods like NER, noun-phrase extraction or even a model that is pre-trained on extracting answers from a provided context.

- **End-to-End Question Generation.**

  In end-to-end question generation the model is asked to generate questions without providing the answers.

  In general, we use models that are pre-trained or fine-tuned to do this task.

1.3 text pre-processing

The common part between all textual data processing types is the pre-processing part, which aims mainly to transform this kind of data into a machine-readable form, and we can distinguish between two types of pre-processing, Classical, and modern. To achieve our objective, we will base ourselves on the modern methods and we will explain why in the next step.

It is only after the pre-processing part in an NLP-oriented project that we can start to adopt techniques that are 100% oriented towards the main objective of the project (Long Form question answering and Question generation in our case).

So, we can distinguish between two types of pre-processing of textual data: classic and modern.

1.3.1 classic text pre-processing techniques

Classical methods are methods that have been mainly based on the number or frequency of occurrence of a word or n-gram - n-gram: sentence of n words - in a document to apply the transformation of the text into a machine-readable vector. Among the most used classical methods are "**Count Vectorizer**" and "**Term-frequency, Inverse Term Frequency Vectorizer. TF-IDF**". Figure 1 is an example of the application of count Vectorizer on some sentences.

Tokens Sentences	The	football	red	great	colour	is	a	NLP
Football is great	0	1	0	1	0	1	0	0
Red is a colour	0	0	1	0	1	1	1	0
NLP is great	0	0	0	1	0	1	0	1

Figure 1: Count_vectorizer Example

1.3.2 modern text pre-processing techniques

In contrast to classical methods, modern methods are mainly an Artificial neural network (**ANN**) which aim to process sequential data, in our case we will focus on Transformers.

The first difference between modern and classic text pre-processing methods is that in the case of the first one, the semantic part of the words is taken into consideration during the transformation, furthermore modern methods have a neural network architecture which is one of the reasons why this kind of pre-processing performs better than classic methods, especially for the processing of massive or complicated textual data.

Considering the modern methods, we can find different sub-categories:

- **Context Independent word embedding methods.**

Context independent models don’t take in consideration the context of the word during the embedding process, that means even if a word is used in two different contexts it will have the same representation.

For example if we consider the sentence «**He brought his cell phone to the prison cell** », In this case although the word cell is used twice in a totally different contexts, the representation of this word will be the same if we use a context independent méthode. And among the context independent embedding methods there's Glove and word2vec.

Figure 2 is an example of the representation of the following words, woman, man, queen, and based on the word2vec representations we can see that:

$$\boxed{King = Queen - woman + man}$$

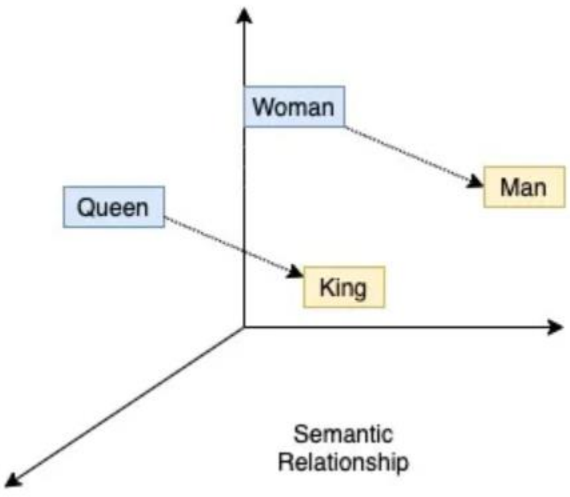


Figure 2: Example of word representation by word2vec

- **Context Sensitive word embedding methods.**

Context sensitive word embedding methods are one of the newest methods in NLP, and they have already showed it capabilities in the NLP, and one of the examples that shows how good context Sensitive are, is that the latest model developed Open AI « **Chatgpt** » uses this kind of methods.

The main idea of this kind of methods is that the context of the word is taken in consideration while the embedding process, which means that if we took the example, we used previously we will have different representations for the same word « **cell**», based on its context in the sentence.

This kind of embedding is generally done by seq2seq like RNN and LSTM models and transformers like Bert and T5.

We will go deeper about the context dependent embedding methods in the next sections.

1.4 Language Models

A language model is a type of artificial intelligence model that is trained to predict the next word or sequence of words in a given text. Language models are used in a variety of applications, such as machine translation, text generation, question generation.

Statistical language models are based on The Markov Assumption, which says that the $K + 1$ depends on the previous k words.

There are different types of language models:

- **Statistical language models**

Statistical language models use probabilistic techniques to predict the likelihood of a sequence of words based on their statistical patterns. In other words, statistical models are able to predict the next word in a sentence based on the words that precede it.

And they are used generally for purposes like speech recognition which is the engine to the voice assistants like google assistant ad Siri, Machine Translation, text Suggestions which we can find in Gmail, Facebook and Microsoft Word.

Generally, we can differentiate between three types of statistical language models:

- **N-gram**

This model considers sequences of n words (also called n-grams) instead of individual words to predict the next word in a sequence. The idea is that the probability of a word is dependent on the previous n-1 words. The larger the value of n, the more context is considered.

- **Exponential**

This model can also be called as MaxEnt model, and its objective is to assign the maximum entropy to the predicted values in order to minimize the statistical assumptions that can be made. In other words, MaxEnt models aim to find the probability distribution that maximizes the entropy while still fitting the observed data.

the entropy is defined over sequences of words. The features and parameters of the desired results are given by the frequency of word n-grams in the training corpus.

- **Continuous space**

This type of model represents words as a non-linear combination of weights in an ANN, which means it performs word embedding. This type becomes especially useful as data sets become larger or when it contains numerous unique words.

The presence of a lot of unique or rarely used words can cause problems for linear model like an n-gram. This is because the amount of possible word sequences increases, and the patterns that inform results become weaker. By weighting words in a non-linear, distributed way, the continuous space model can "learn" to approximate words and therefore not be misled by any unknown values. Its understanding of a given word is not as tightly tethered to the immediate surrounding words as it is in n-gram models.

But Despite being able to perform on many tasks as LMs, this models have shown some drawbacks too, firstly the zero probabilities issue that appears when a vocabulary in the input data isn't in the model's vocabulary as a result it's assigned with a zero probability which leads to an infinite perplexity. The second problem is the exponential growth of n-grams when we increase the value of n. And the last issue of SLMs is the lack of generalization.

- **Transformer-based language models**

  The transformer-based language models uses the architecture partially or completely of a transformer like Bert, T5, and GPT in order to be able to use the self-attention and the attention mechanisms which allow the model to determine the importance of each word when predicting the next word in the output sentence.

1.5 Sequence to sequence models

Seq2sec models are simply a kind of ANNs that takes a sequence of words like a sentence or text an input and generates another sequence of text as an output.

We can mention a lot of examples of the application of seq2seq models:

- **Text generation**
- **Text summarization**
- **Question generation**
- **Translation**

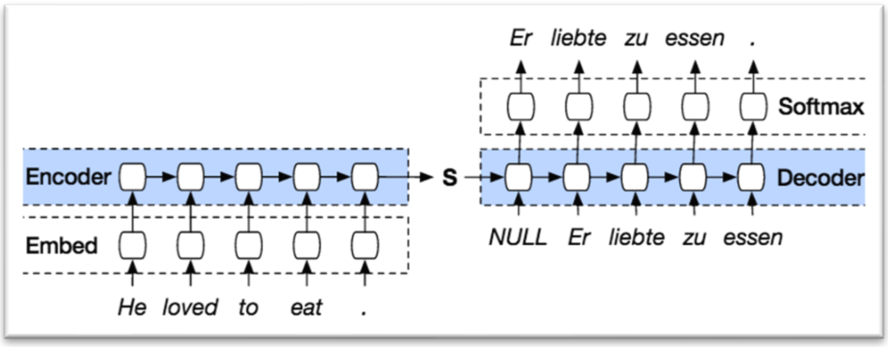


Figure 3: seq2seq model Example

1.5.1 Recurrent Neural Networks (RNNs)

Recurrent Neural networks are a type of ANNs that can be trained on sequential data based on BPTT or the RTRL approach in order to generate sequences as an output. And they are Widley used in NLP, speech recognition, time series…

One of the advantages of RNNs is that it doesn't have a fixed input size, just like in a translation tool for example we can provide sentences with different sizes and the model will still be able to translate all of them to the target language.

Furthermore, The RNNs are distinguished by their memory that allows them to guarantee a better processing of the input sequence by taking information from the prior inputs to influence the current input and output. As we can see in figure 4, each state in our RNN depends on the previous state and the current input.

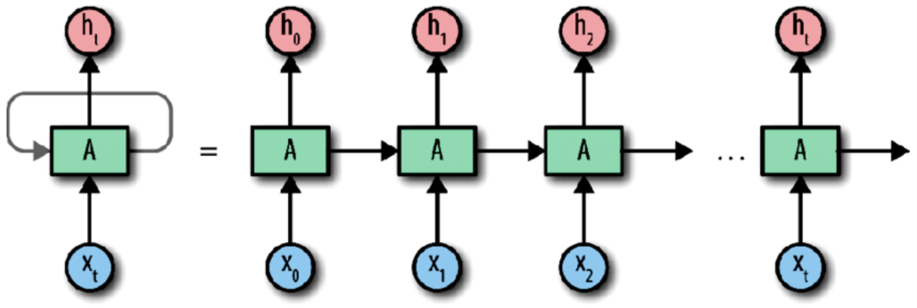


Figure 4: RNN Architecture

And as shown in figure 5, we can differentiate between 4 types of RNNs.

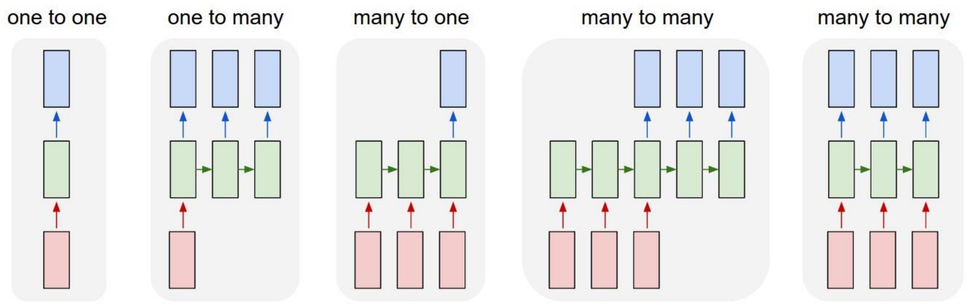


Figure 5: Types of RNNs

But even with all the services that they provide, RNNs weren't able to perform well when dealing with large sequences, because the longer the sequence the more information that need to be stored. And the other significant problem with RNNs is the exploding, vanishing gradient problem.

This problem occurs during the training of the RNN, and exactly when the gradients computed during the backpropagation become very small in the case of the vanishing gradient, or very big in case of the exploding gradient which might lead to a very slow process in order to update the weights of the network and in some cases to a total failure to converge.

In order to overcome these problems many methods can be used like Gradient clipping, Reducing The number of layers can help us but we will be giving up some of our model's complexity, or we can replace our RNN by an LSTM.

1.5.2 Long-short term memory (LSTM)

LSTM is type of RNNs architecture specifically designed to address the issue of long-term dependencies in traditional RNNs. LSTMs are well suited for tasks such as sequential prediction, where the network must consider information from earlier timesteps in order to make a prediction. The key innovation of the LSTM architecture is the introduction of gating mechanisms as shown in figure 6, which allows the network to control the flow of information and preserve long-term dependencies.

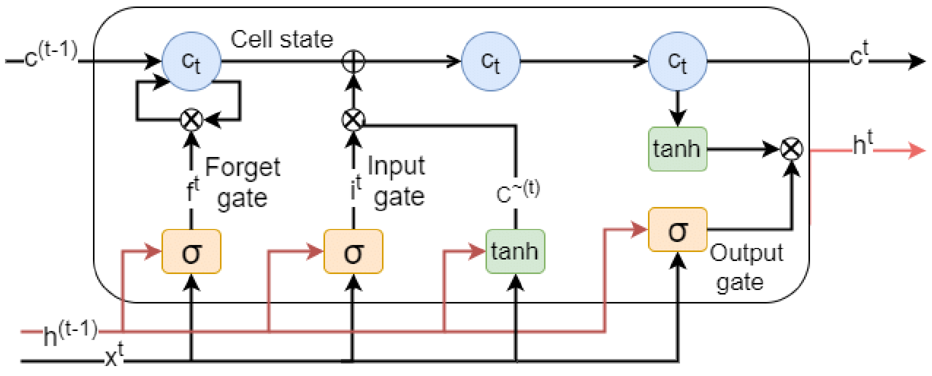


Figure 6: The architecture of LSTM

LSTMs overcome the vanishing and exploding gradient problem by using the forget gate, the input gate, and the output gate within their architecture to selectively retain or discard information.

These gates can regulate the flow of information in the network and prevent gradients from becoming too large or too small. Additionally, the use of a gated memory cell helps to maintain long-term dependencies, allowing the network to better preserve information from earlier timesteps that would otherwise be lost with traditional RNNs.

In addition to the one-way LSTM, we can only consider the Bidirectional LSTM, that instead of considering only the previous words in the input sequence it considers the next words too, which guarantees a better understanding and better representations by the model.

1.5.3 Gated recurrent Unit (GRU)

Just like the LSTM the GRU can solve the short-term memory issue of the RNN. But instead of using a cell state to regulate information, it uses hidden states. Moreover, instead of three gates, it uses only two: a reset gate and an update gate. Similar to the gates in LSTM, reset gates and update gates control the amount of information and which one to be retained.

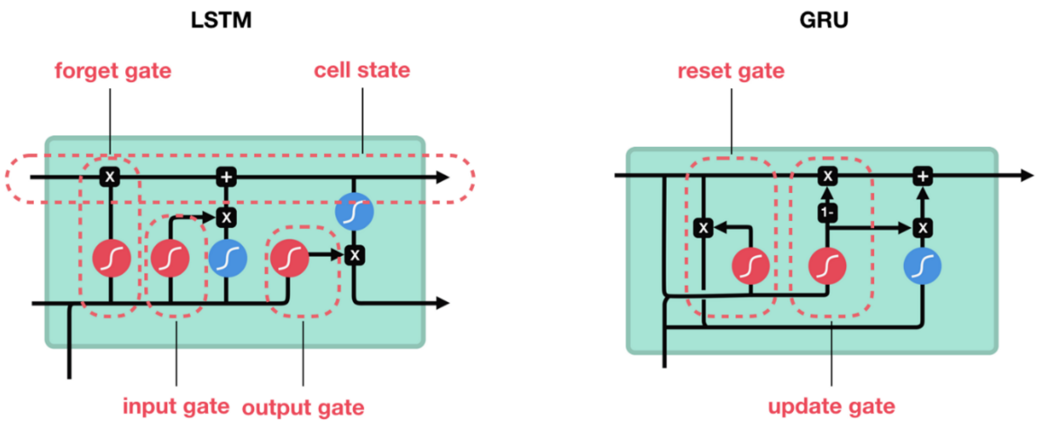


Figure 7: Differences between the architectures of LSTM and GRU

1.5.4 The Encoder-Decoder Architecture

As we can see in figure 8, The encoder decoder is combination of two neural networks, that usually have the same architecture, but different functionalities such as the first one that can be referred to as the encoder is used to convert the input sequence into a vector of numbers that represents the context of our input, while the second on is supposed to do the opposite by converting the context vector generated by the encoder into an output sequence.

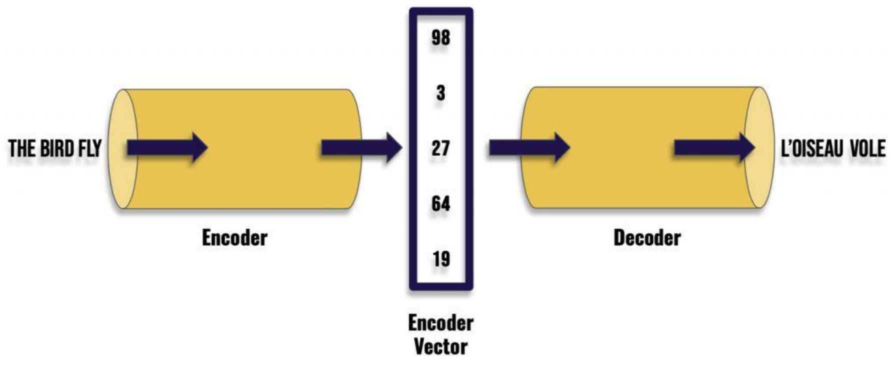


Figure 8: The encoder-decoder architecture

The main advantage of using encoder-decoder to perform this kind of tasks instead of processing word by word is that while using this architecture we are considering the global meaning of the sentence, which means that we consider the dependencies between the words, which guarantees a better understanding of the sequence. Which is not the case when we adopt a word-by-word processing.

In figure 9 we can see that the encoder and the decoder are generally a sequence of RNNs, LSTMs ore GRUs that takes a word as an input, collects information from it by calculating its hidden state based on the current word and the previous hidden state using the following formula.

$$h_t = f(W^{(hh)}h_{t-1} + W^{(hx)}x_t)$$

And after representing the whole sequence the decoder receives the hidden vectors and starts generating the output and updating its hidden vectors based on the previous ones following this formula.

$$h_t = f(W^{(hh)}h_{t-1})$$

And in order to determine the output words y_t we use a SoftMax in order to calculate a probability vector which helps to decide the final output from the model's vocabulary.

$h_t: current\ hidden\ state.$

$h_{t-1}: previuos\ hidden\ state.$

$x_t: current\ input\ word.$

$W^{(hh)}: weight\ matrix\ of\ size\ h \times h.$

$W^{(hx)}x_t: weight\ matrix\ of\ size\ h\ \times x.$

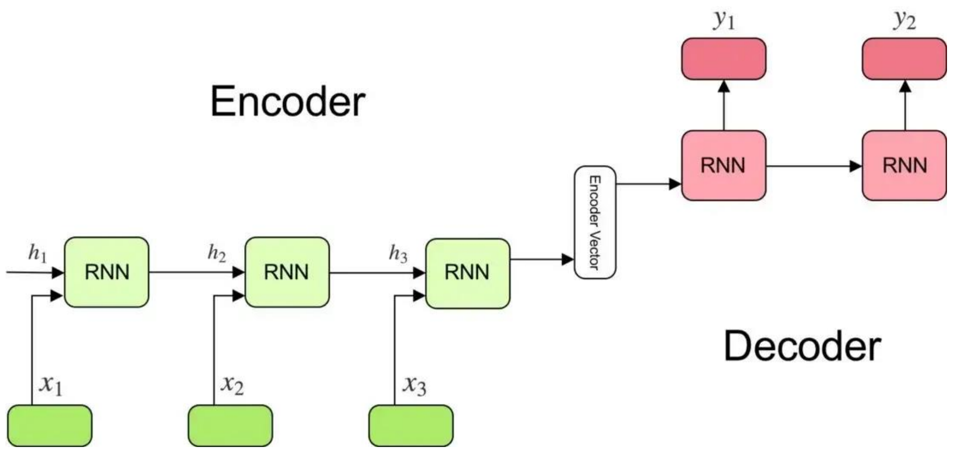


Figure 9: detailed encoder-decoder architecture

Other than the inherited advantage of being able to process different input sizes, this simple encoder-decoder can be significantly improved by adding some mechanisms like the attention and the self-attention that makes it possible to capture more information by using multiple encoder vectors instead of only one vector, which means that processing long and complex sequences won't be a very big problem. because attention is all you need.

1.6 Transformers

As we mentioned earlier Natural Language processing, is one of the biggest branches of Machine / deep learning, and it has many beneficial applications like sentiment analysis, Question answering, Text summarization …

That’s why there was a continues research done to improve the performance of the tools we use to perform NLP tasks, and the latest results of this research are the transformers, which have shown a very big improvement in the field of NLP, especially when dealing with complex data or complex tasks that the old methods like RNNs, LSTMs were not able to perform perfectly. Let’s discover how and why transformers can provide a such big improvement in the NLP field.

1.6.1 Definitions

Transformers are an encoder-decoder architecture that we can use partially or completely to perform an **NLP** task, the architecture of a transformer has three main parts:

- **Encoding part**
  - This is the part of the transformer that receives the inputs (the inputs are generally word embeddings with positional encoding in order to give an information about the position of each word, or the distance between different words in the input data).
  - This part is composed by a group of encoders which are the main components of the encoding parts. And here is the architecture of an encoder.

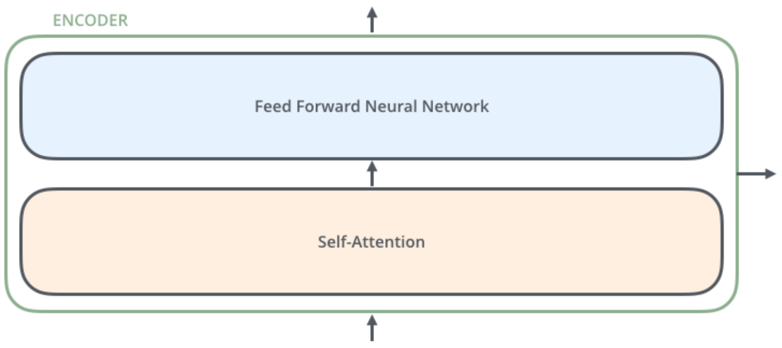


Figure 10: The architecture of an encoder

  - **The output** of the encoding part is a Representation vector and self-attention vectors for each input.

- **Decoding part**

  - This part of the transformer is responsible of to convert the knowledge (Embeddings and Self-Attention vectors) built by the encoders in the first part into an output sequence, and to guarantee an efficient processing we also use the **attention** mechanism.

  - This part is composed by a group of decoders that have the following architecture:

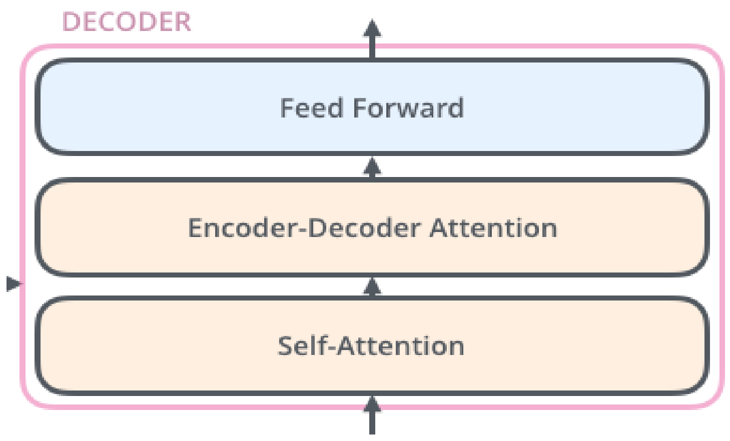


Figure 11: The architecture of Decoder

  - The output of the decoder is the targeted sequence (For example if we are performing translation the output of the decoder will be the sentence in the targeted language).

1.6.2 encoder Decoder Architecture: Definition

- Encoding Part

1.6.2.1 Self-attention

➢ **What is self-attention?**

Self-attention is an attention mechanism that allows our model to better understand the and relate between the tokens of the input.

That means that while processing each word of the input sentence, Self-attention allows it to take into consideration the other words to extract knowledge that can lead to a better encoding to the processed word.

In this example we can see the application of self-attention while processing the word the as the clearest the line, the bigger the relation between the two words.

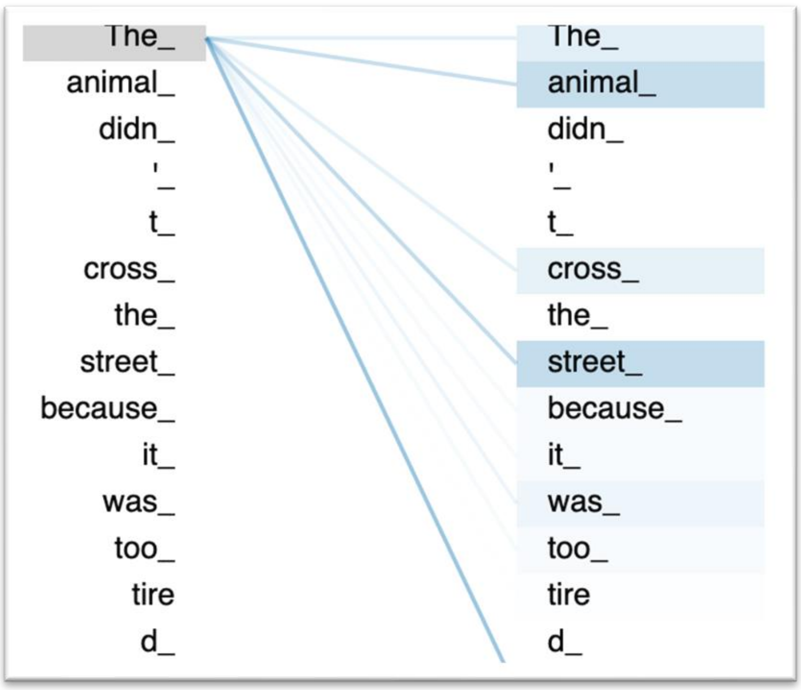


Figure 12: Self-attention Example.

How does self-attention work?

In order to better understand how the self-attention works, we will break it down into five steps:

- **Calculating The Query, Value and Key vectors**

Firstly, the model receives a given number of vectors (embeddings of the tokens of the input sentence plus positional encodings), so the first step on the self-attention layer is to calculate three **-very important-** vectors.

To simplify the explanation let's consider $\boldsymbol{X_i}$ as the embedding of the word that we are encoding.

- **The query vector (Q)**

This vector represents what kind of information we are looking for, and it's calculated by multiplying the vector X **by** a weights matrix That we can call $\boldsymbol{W^Q}$**.**

$$\left(x_{1,}\ x_2, x_{3,}\ \ldots.,x_N\right)\ \times \begin{pmatrix} W^Q_{(1,1)} & \cdots & W^Q_{(1,M)} \\ \vdots & \ddots & \vdots \\ W^Q_{(N,1)} & \cdots & W^Q_{(N,M)} \end{pmatrix} = \left(q_{1,}\ q_2, q_{3,}\ \ldots.,q_M\right)$$

Figure 13: Calculating the query vector.

- **The key vector (K)**

This vector represents the relevance to the query, and it's calculated by multiplying the vector X **by** a weights matrix That we can call $\boldsymbol{W^K}$**.**

$$\left(x_{1,}\ x_2, x_{3,}\ \ldots.,x_N\right)\ \times \begin{pmatrix} W^K_{(1,1)} & \cdots & W^K_{(1,M)} \\ \vdots & \ddots & \vdots \\ W^K_{(N,1)} & \cdots & W^K_{(N,M)} \end{pmatrix} = \left(k_{1,}\ k_2, k_{3,}\ \ldots.,k_M\right)$$

Figure 14:Calculating the Key Vector.

- **The value vector(V)**

This vector represents the actual contents of the input and it's calculated by multiplying the vector $\boldsymbol{X_i}$ **by** a weights matrix That we can call $\boldsymbol{W^V}$**.**

$$\left(x_{1,}\ x_2, x_{3,}\ \ldots.,x_N\right)\ \times \begin{pmatrix} W^Q_{(1,1)} & \cdots & W^Q_{(1,M)} \\ \vdots & \ddots & \vdots \\ W^Q_{(N,1)} & \cdots & W^Q_{(N,M)} \end{pmatrix} = \left(v_{1,}\ v_2, v_{3,}\ \ldots.,v_N\right)$$

Figure 15: Calculating the value vector.

Of course, we should take in consideration that $W^V, W^V., W^K$ *are from* $R^{N\times M}$*, with N is the dimension of the embedding of the input vectors.*

- ## Calculating the dot product of Q and K^T

After calculating the key vector, query vector, and the value vector of each token, we can now start calculating our Attention values.

First, we will consider two matrices:

Q: The i_{th} row of this matrix is the query vector of the i_{th} word in the input sequence.

K: The i_{th} row of this matrix is the key vector of the i_{th} word in the input sequence.

We calculate the dot product $\boldsymbol{Q} \times \boldsymbol{K^T}$, that product will result us a $m \times m$ Matrix that represents the attention scores among the input tokens. $(M \text{ is the number of words in our sequence})$

Note: it's because of the physical meaning of the dot product why we can say that the resulted matrix represents the attention scores among the tokens.

- ## The division on the dimension of the key vectors

Let d_k the dimension of our key vectors.

In this step we will divide our previous product by the square root of d_k

Our formula now is $\dfrac{\boldsymbol{Q \times K^T}}{\sqrt{\boldsymbol{d_k}}}$

- ## Scaling the matrix values by SoftMax

In this step we will simply apply a SoftMax on our resulted matrix so far in order to scale its values. $\boldsymbol{softmax(\frac{Q \times K^T}{\sqrt{d_k}})}$

and we have $Softmax(X_i) = \dfrac{e^{X_i}}{\sum_{i=0}^{n} e^{X_i}}$

- ## Calculating the self-attention vectors

the last step of calculating our self-attention vectors is to multiply the matrix we have so far by a matrix **V The $\boldsymbol{i_{th}}$ row of this matrix is the value vector of the $\boldsymbol{i_{th}}$ word in**

the input sequence, this step will allow us to get an adjusted embeddings of the tokens because as we mentioned before the V vector represents the content of each token in the input sentence.

$$Self\_attention = \boldsymbol{softmax}\left(\frac{\boldsymbol{Q} \times \boldsymbol{K}^{\boldsymbol{T}}}{\sqrt{\boldsymbol{d_k}}}\right) \times \boldsymbol{V}$$

1.6.2.1 Feed-forward layer

The role of this layer is generally to transform the data into a suitable form for the subsequent layers to process, in an encoder we can use it to reduce the dimensionality or to extract the features.

▪ Decoding part

As we saw earlier the decoder has three layers, a self-attention layer, an encoder-decoder attention layer and a feed-forward layer.

1.6.2.2 self-attention layer (Decoder)

In the decoder we calculate the self-attention for the output vector, but the difference this time is that this layer has access to only the earlier tokens of the output vector, and that is done by setting the previous positions to $-\infty$ before calculating the SoftMax**.**

The objective of putting this layer is to help the model to pay attention to the output sequence, because while generating the next word in the output sentence we should have an idea about the words we are having so far.

1.6.2.3 The encoder-decoder layer

The objective of this layer is to apply the attention mechanism while generating the output sequence, and the main idea behind this mechanism is to solve one of the problems that we had with RNNs, which is the inability to perform well while dealing with long sequences, because the way it works lead to losing information because it tries to represent the context of the whole sequence in one and only hidden state vector.

In order to solve this problem here is the steps that we do while performing the attention mechanism on a sequence-to-sequence model.

- **The first step**

  The Encoder-Decoder attention layer receives a set of attention vectors (**Only Key and Value vectors**) which are the output of the last encoder in the decoding part, in addition to an initial decoder hidden state.

- **The second step**

  Here the decoder or specifically the encoder-decoder layer processes the input (**Key vector, Value vector and a decoder hidden state vector $\boldsymbol{h_i}$**), and it produces a representation of the output word and a new decoder hidden state vector $\boldsymbol{h_{i+1}}$**.**

- **The third step**

  This is the step where we can notice and understand how the attention really works, because we use the inputs generated by the encoder, and the decoder hidden state $\boldsymbol{h_{i+1}}$ **to** generate a context vector C_{i+1}.

  The dimension of the vector C_{i+1} is the number of the words in the input sequence (which is also the number of the representation vectors we get from the encoding part)

  The values that we have in the vector C_{i+1} represents the weight between each word in the input sentence and the one that is being processed.

  for words that the model should pay attention to while processing the current word w_i will have weight $\alpha_i \approx 1$, while the words that are not related to w_i would have a weight $\alpha_i \approx 0$

- **The fourth step**

  We concatenate the vector C_{i+1} and h_{i+1}in order to pass the generated vector into a feed-forward neural network that will result the output word. this whole process repeated until the output sequence is fully generated.

1.6.2.4 feed-forward layer(decoder)

The role of this layer is generally to transform the data into a suitable form for the subsequent layers to process, in an encoder we can use it to reduce the dimensionality or to extract the features, and the output of the feedforward layer is the last representations of the words of our output sequence.

1.6.2.5 The last part of the transformer

This is the part where we convert the representations of the output sequence words into real words that we can understand, and here is its architecture.

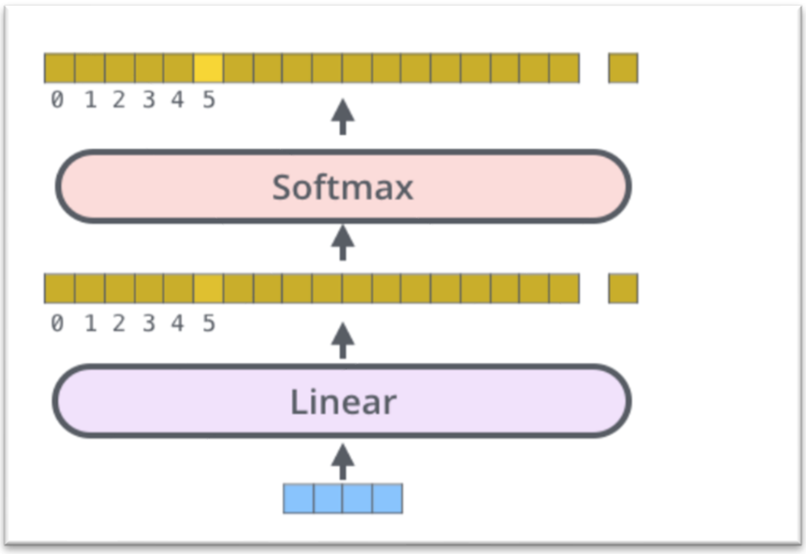


Figure 16: The last part of the transformer

1.6.2.6. The linear layer

The role of this Layer is to convert each representation generated by the decoder into a logits vector.

The dimension of the logits vector equals to the number of the words that our model knows (the words that our model was trained on), and each cell in our logits vector will contain the score of a word in our model's vocabulary.

1.6.2.6 SoftMax Layer

After generating the logits vector, we use a SoftMax function to turn our logits vector into a vector of probabilities that adds up to one and of course our output word will be the word that corresponds to the cell with highest probability.

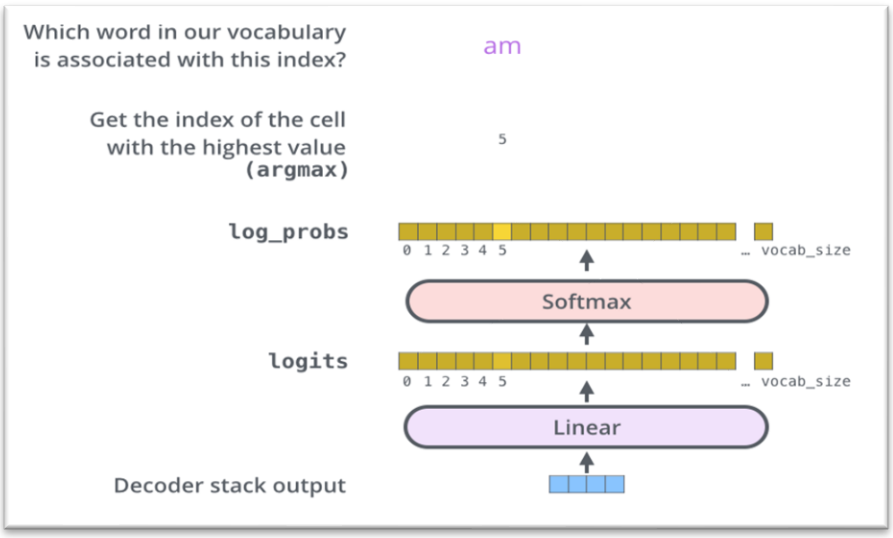

Which word in our vocabulary is associated with this index?
am
Get the index of the cell with the highest value (argmax)
5
log_probs
0 1 2 3 4 5
… vocab_size
Softmax
logits
0 1 2 3 4 5
… vocab_size
Linear
Decoder stack output

Figure 17: Example of the linear and the SoftMax layers functionality

Conclusion

In this section we mentioned the question answering types and methods, the question generation types, and we also discussed and explained the notions that can be used in order to achieve the objectives of our project, like the language models that can be the first option when the objective is to generate text, and sequence to sequence models that are used generally to in translation and question answering. And we finished by a detailed explanation of the transformers that we will be using in this project. (Introduction to the next chap)

2 Methodology

In this section we will be presenting the methods and the approaches that we adopted in order to achieve the objectives of this project which are:

- Text extraction from a document.
- Generate the embeddings of the extracted text.
- Store the embeddings in pinecone which is an online vector database.
- End to End Question Generation based on the extracted text.
- Perform question answering on the given question based on the given context and also the documents that we are storing.

2.1 Extracting text from a document.

The first task that our application should be able to fulfil Is to extract text from a document, the first version of documents that we will be working on is PDFs.

2.1.1. PyPDF2

PyPDF2 is a free and open-source pure-python PDF library capable of splitting, merging, cropping, and transforming the pages of PDF files. It can also add custom data, viewing options, and passwords to PDF files. PyPDF2 can retrieve text and metadata from PDFs as well.

Figure 18: PyPDF2 LOGO

The main PyPDF2 functionality that we used in this project is extracting text from PDF pages.

2.2 Store the embeddings in pinecone.

2.2.1. Pinecone

Pinecone is a vector database that makes it easy for developers to add vector-search features to their applications, using just an API.

Vector databases are purpose-built to handle the unique structure of vector embeddings. They index vectors for easy search and retrieval by comparing values and finding those that are most similar to one another. They are, however, difficult to implement.

Figure 19: Pinecone logo

In our case we will be using pinecone in order to keep the word embeddings that we generate each time a user uploads a document.

The main objective of keeping the embeddings of the previously generated embeddings is to enrich the source that we will be relying on to answer future questions.

And in order to avoid any kind of misleading answers, pinecone offers a méthode to calculate the similarity between the given question and the stored vectors, so we can choose the number stored documents to rely on based on this parameter.

This feature can also give us the ability to answer a question even if the answer isn’t in the provided document.

2.3 Question Answering and question generation.

2.3.1 Transformers

The transformers library in Python is a tool that enables you to utilize pre-trained transformer models for various NLP tasks. It provides a user-friendly API for working with models such as BERT, GPT, and RoBERTa. It also allows for fine-tuning the models with custom data and training new models. Furthermore, this library supports multiple deep learning frameworks including PyTorch and TensorFlow, which makes it easy to integrate with existing projects.

Figure 20: Transformers Logo

In this project we will be using Transformers pre-trained models in order to perform text embedding, question answering and question generation.

The architectures that we used in this project are:

- **Bert**

BERT is a transformer-based deep learning model developed by Google that can be used for NLP tasks such as language translation and question answering ...

One of the amazing characteristics of BERT is that it is bidirectional, which means that it considers the context on both the left and right sides of each word in a sentence. That makes BERT able to understand the meaning of words in sentences deeper than the classic models that capture the context on one side.

- **Pre-training**: Bert was pre-trained using a large dataset retrieved from various sources like, Wikipedia, and Book Corpus. Being trained on a such big dataset allows BERT to learn general language patterns and structures that can be useful for a variety of NLP tasks.

- **Pre-training Objectives: Bert** has two main pre-training objectives that were used to train that model, to allow it to learn general language patterns and structures. Those pre-training objectives are the following:

    - **Masked language modeling (MLM)**: In this objective, the model is trained to predict the value of masked tokens in a sequence of words based on the context provided by the other words in the sequence.

    - **Next sentence prediction:** In this objective, the model is trained to predict whether a given sentence follows a given sentence in a document. This helps the model understand the relationships between sentences and the context in which they appear, which can be useful for tasks such as **question answering** and text summarization.

- **Fine-Tuning:** To use a pre-trained model for a specific task we can fine-tune it by training it on a smaller, task-specific dataset. In our example we can use a long form question answering dataset to fine-tune Bert and then use it to Generate Questions and provide answers based on a provided document.

  The steps to fine-tune The Bert Model are:

    - Pre-process the input data.
    - Define the fine-tuning mode.
    - Train the fine-tuning model.
    - Evaluate the fine-tuned model.

- **Bart**

  BART (Denoising Autoencoder for Pre-training Sequence-to-Sequence Models) is a neural network-based NLP model developed by Facebook AI Research. It is a variation of the transformer architecture.

  BART is pre-trained on a massive amount of text data and can then be fine-tuned for a wide range of NLP tasks such as text generation, text summarization, and text-to-text transfer. The model is trained on a denoising autoencoder-based pre-training task, where the model is trained to reconstruct a corrupted version of the input text. This pre-training allows the model to learn a wide range of patterns and relationships in language, and the denoising aspect of the pre-training task helps the model to learn to handle noise and other types of errors in the input.

  After pre-training, the model can be fine-tuned for different tasks such as text summarization and text-to-text transfer. The fine-tuning process involves training the model on a small amount of task-specific data and fine-tuning the model's parameters to better perform the specific task.

- **T5:**

  T5 uses a transformer architecture, which is based on the idea of self-attention. In a transformer-based model, each word in the input text is represented by a vector, and the model learns to attend to certain parts of the input text when generating the output. The transformer architecture also includes multi-head attention, which allows the model to attend to different parts of the input text at different levels of granularity.

  One of the key features of T5 is its pre-training on a massive amount of text data, which allows it to learn a wide range of patterns and relationships in language. This pre-training enables T5 to perform well on a wide range of NLP tasks without the need for task-specific fine-tuning.

  T5 is trained using a task-agnostic training regime, which means it is trained to generate text for any task, given a prompt that describes the task. This allows for a more efficient and flexible use of the model, as it can be fine-tuned for a new task using much less data than would be required to train a model from scratch.

- **All-mpnet-base-v1:**

  This is a sentence-transformers model, it maps sentences and paragraphs to a 768-dimensional dense vector space and can be used for tasks like clustering or semantic search.

  This model is intended to generate output vectors that represents the semantic information in the input sequence. The output vectors can be used for many tasks like information retrieval, clustering, or similarity task.

  In order to build this model, the Microsoft/mpnet-base is used as a pretrained model, then it’s fine-tuned using a concatenation from multiple datasets with a total number of 1 billion sentences, by computing the cosine similarity from each possible sentence from the batch, then a cross entropy loss is applied by comparing with the true pairs.

2.3.2. Haystack

Haystack is a Python library that allows for easy and efficient search and retrieval of information from large datasets. It provides a simple and consistent API for interacting with various search backends, such as Elasticsearch.

Haystack can be used to search for and retrieve specific pieces of information from a dataset, as well as to perform advanced search queries and aggregations.

Figure 21: Haystack logo

Haystack also provides a simple way to add search functionality to your Django or Flask application. It includes built-in support for indexing and searching models, as well as support for custom indexing and search logic. The package provides a SearchQuerySet API that allows you to perform complex search queries and retrieve the results in a format that's easy to work with in your application. Additionally, it provides a Search Index class that you can use to customize how your models are indexed and searched. With Haystack, you can easily add powerful search functionality to your application, without having to spend a lot of time and effort working with the underlying search engine.

2.3.3 PyTorch

PyTorch is a powerful, flexible, and user-friendly deep learning library for Python. It is designed to be easy to use and intuitive for practitioners, while also being flexible and extensible for researchers. Some of the key features of Pytorch include:

Dynamic computational graph: Unlike other deep learning libraries, PyTorch uses a dynamic computational graph, which allows for more flexibility and faster model experimentation. This means that the user can change the graph on the fly, during the execution of a program, without the need to rebuild the graph.

Automatic differentiation: Pytorch provides automatic differentiation capabilities, which allow for efficient computation of gradients during the training process. This means that the user does not need to manually specify the gradients, which can save a lot of time and effort.

CUDA support: PyTorch is built to take advantage of CUDA-enabled GPUs, which can significantly speed up the training process for large models.

Interoperability: PyTorch is designed to be easily integrated with other libraries and frameworks, such as TensorFlow and scikit-learn. This allows for the use of PyTorch in a variety of different environments and workflows.

Figure 22:Pytorch Logo

2.3.4 natural language toolkit (NLTK)

The Natural Language Toolkit is a library in Python that provides tools to work with human language data. It includes modules for tasks such as tokenization, part-of-speech tagging, sentiment analysis, and more. NLTK is widely used for teaching, research, and development in the field of NLP. It also has a large community and extensive documentation, making it a great resource for NLP beginners and experts alike.

Figure 23:NLTK logo

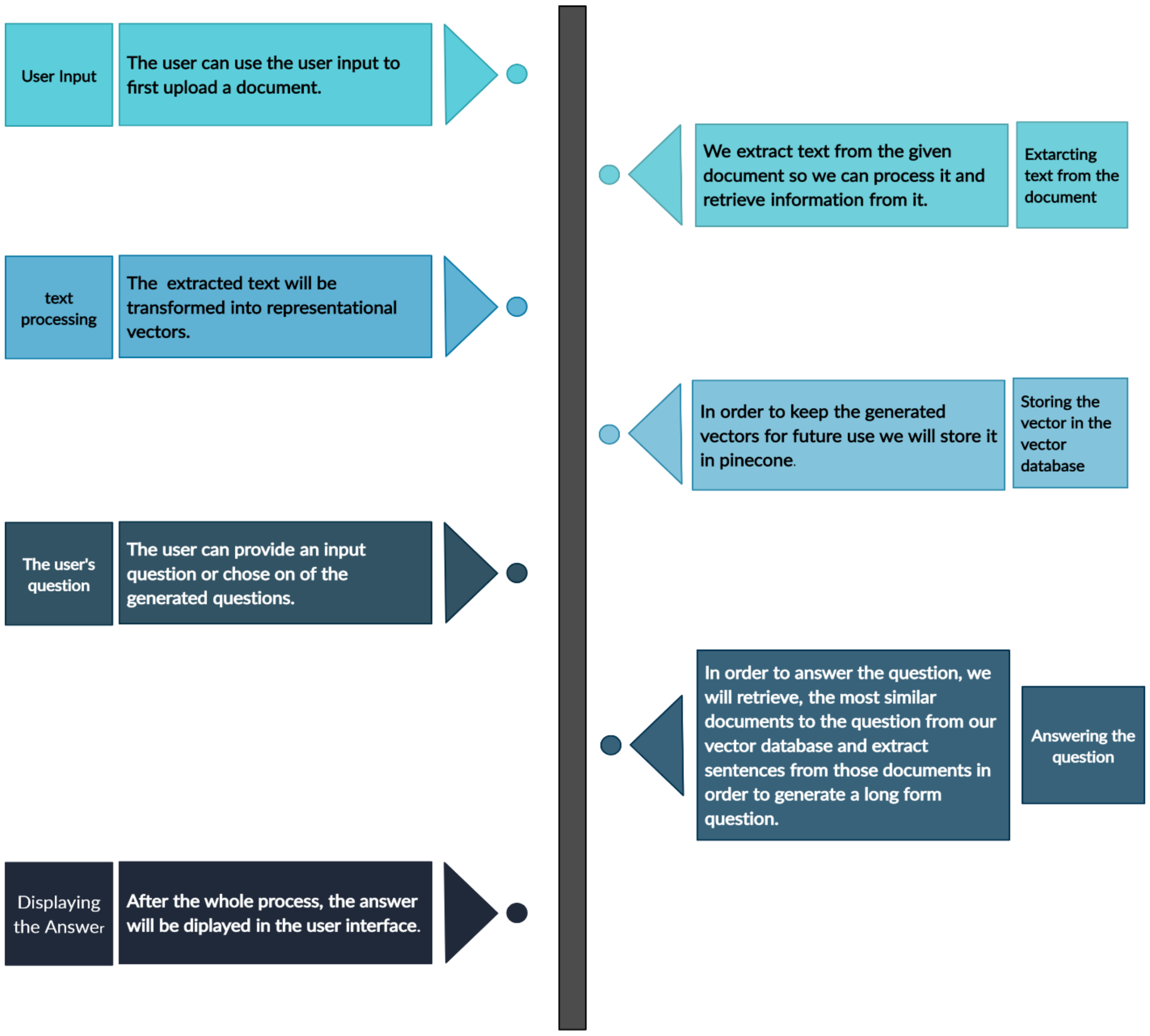
2.4 User Interface

2.4.1 Streamlit

Streamlit is an open-source Python library that allows you to create interactive web-based data visualization and machine learning tools quickly and easily. It allows you to build simple to complex web applications with minimal effort, and its user-friendly interface makes it easy for data scientists and developers to create powerful, custom data visualization tools. With Streamlit, you can create interactive dashboards, data visualization tools, and machine learning apps without the need for web development expertise.

Figure 24: Streamlit Logo

2.5 Project workflow



User Input

The user can use the user input to first upload a document.

Extarcting text from the document

We extract text from the given document so we can process it and retrieve information from it.

text processing

The extracted text will be transformed into representational vectors.

Storing the vector in the vector database

In order to keep the generated vectors for future use we will store it in pinecone.

The user's question

The user can provide an input question or chose on of the generated questions.

Answering the question

In order to answer the question, we will retrieve, the most similar documents to the question from our vector database and extract sentences from those documents in order to generate a long form question.

Displaying the Answer

After the whole process, the answer will be diplayed in the user interface.

Figure 25: Project workflow

In figure 26 we can observe the use case of each library in our project.

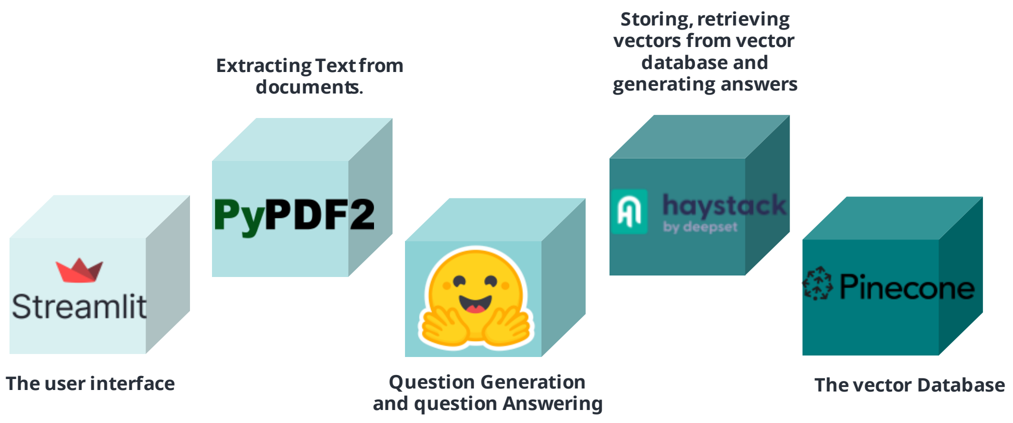
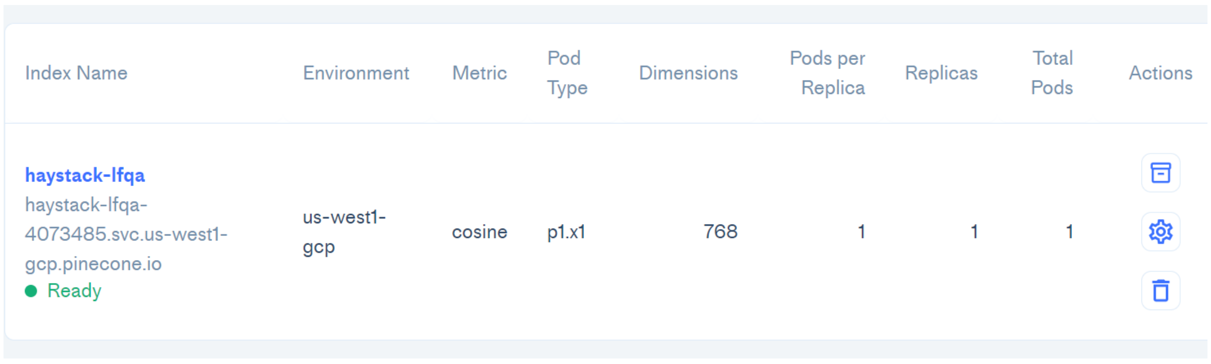

Extracting Text from documents.
Storing, retrieving vectors from vector database and generating answers
Streamlit
PyPDF2
haystack by deepset
Pinecone
The user interface
Question Generation and question Answering
The vector Database

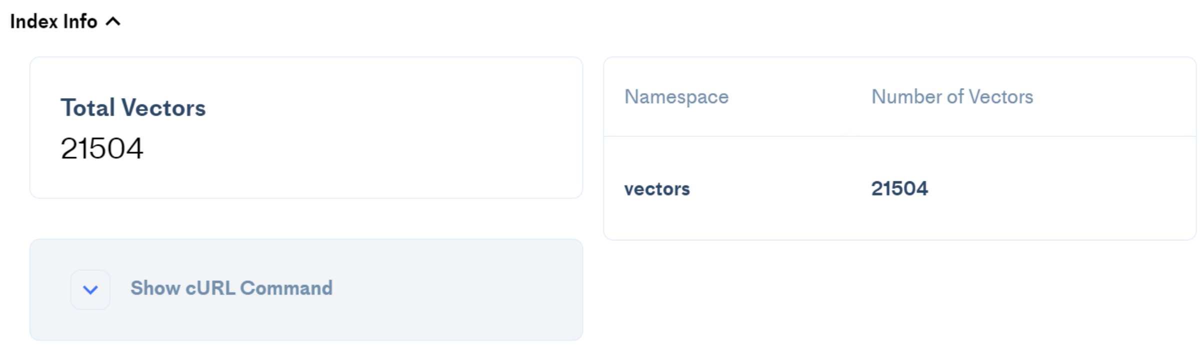
Figure 26: Use cases of libraires

In figure 27 and 28 you can see some information's about our vector database in pinecone.



Index Name	Environment	Metric	Pod Type	Dimensions	Pods per Replica	Replicas	Total Pods	Actions
haystack-lfqa haystack-lfqa-4073485.svc.us-west1-gcp.pinecone.io ● Ready	us-west1-gcp	cosine	p1.x1	768	1	1	1	

Figure 27: Pincone vector data base index



Index Info

Total Vectors
21504

Namespace	Number of Vectors
vectors	21504

Show cURL Command

Figure 28: Information about the vector data base in pinecone

Conclusion

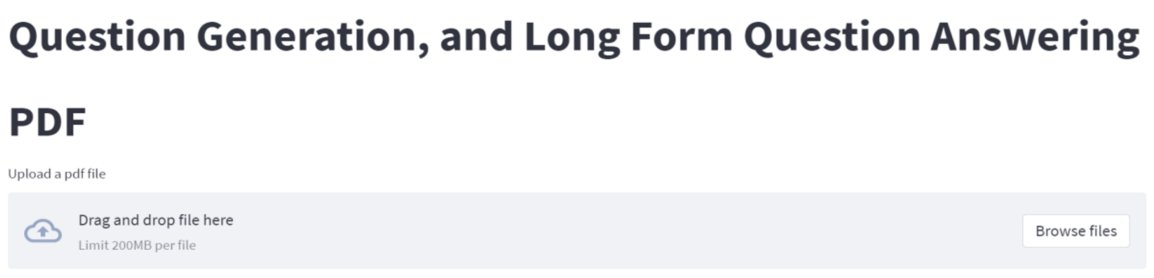
In this section we looked at the tools that we used while building this project, and we tried to clarify the way our model works by constructing a process flow that describes the steps of processing the user inputs until providing the answer, and a simple form that describes the use case of each library in our project. Next, we will see the finale results of our application.

3. Results

In this section we will be presenting the results and the functionalities that we achieved after finishing the development of our application.

In figure 29 we can see the first step of using this application which is uploading the pdf document.

After uploading the pdf document, the process of end-to-end question generation will start, using a t5 architecture model, Valhalla/t5-small-e2e-qg. After that we will use the all-mpnet-base v1 in order to transform the document and add it to our vector database in pinecone.

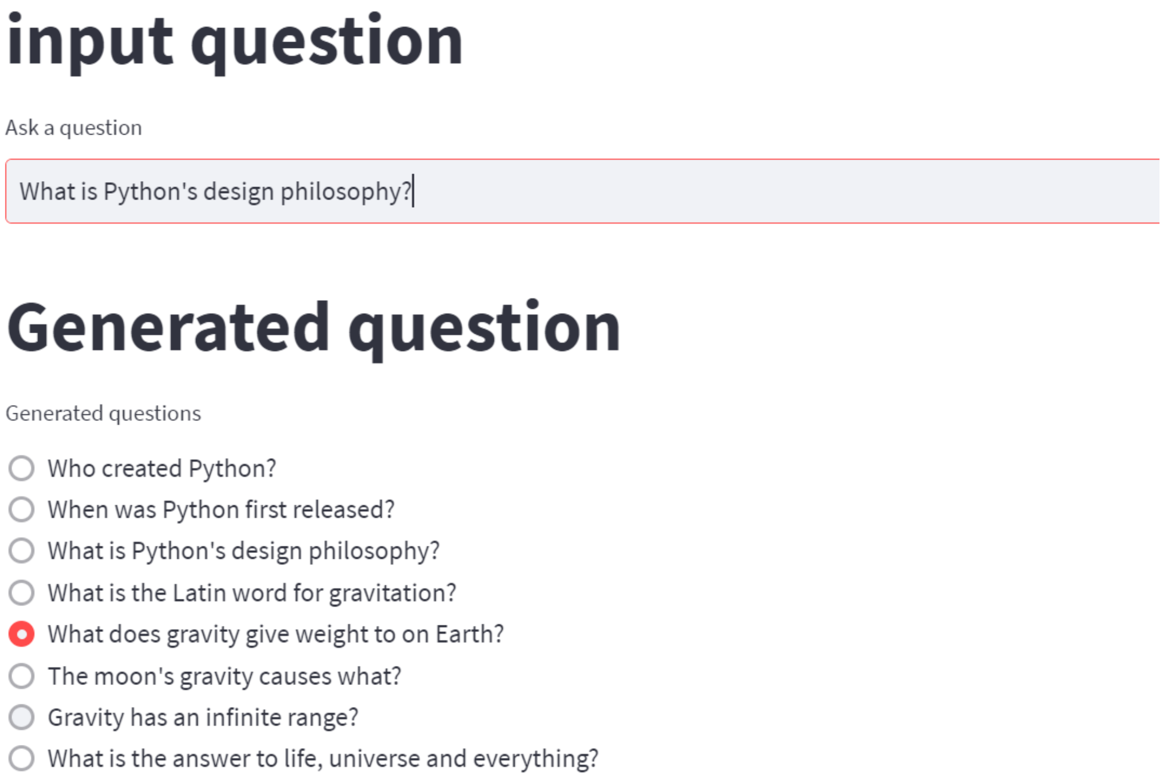

Question Generation, and Long Form Question Answering

PDF

Upload a pdf file

Figure 29: Upload a PDF document.

In figure 30, we can observe both of the ways that the user can submit a question, the first one is by writing his own question as an input, the second one is by choosing of the generated questions.


input question

Ask a question

What is Python's design philosophy?

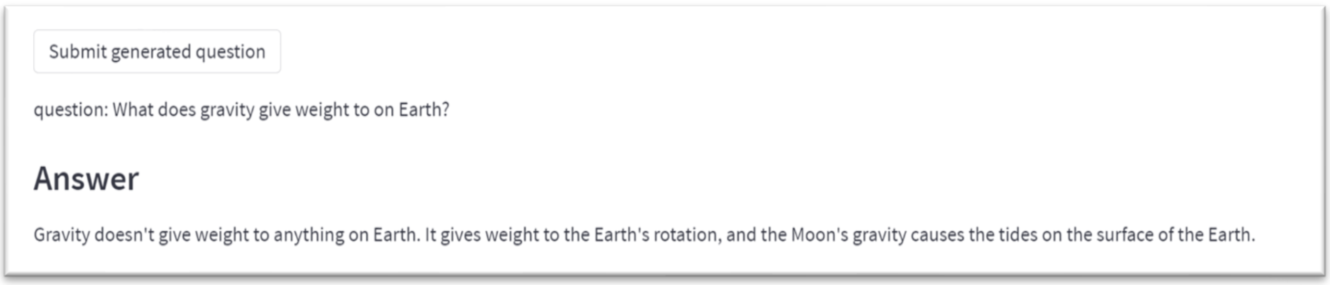
Generated question

Generated questions

- Who created Python?
- When was Python first released?
- What is Python's design philosophy?
- What is the Latin word for gravitation?
- [✓ What does gravity give weight to on Earth?]
- The moon's gravity causes what?
- Gravity has an infinite range?
- What is the answer to life, universe and everything?

Figure 30: user input and generated questions.

In figures 31 and 32, we can see both cases. When we submit a question, a question answering pipeline will start by retrieving the top three similar documents using the sentence-transformer "all_datasets_v3_mpnet-base", then it will start generating the answer based on the retrieved documents using ''vblagoje/bart_lfqa'' which a Bart architecture-based model.

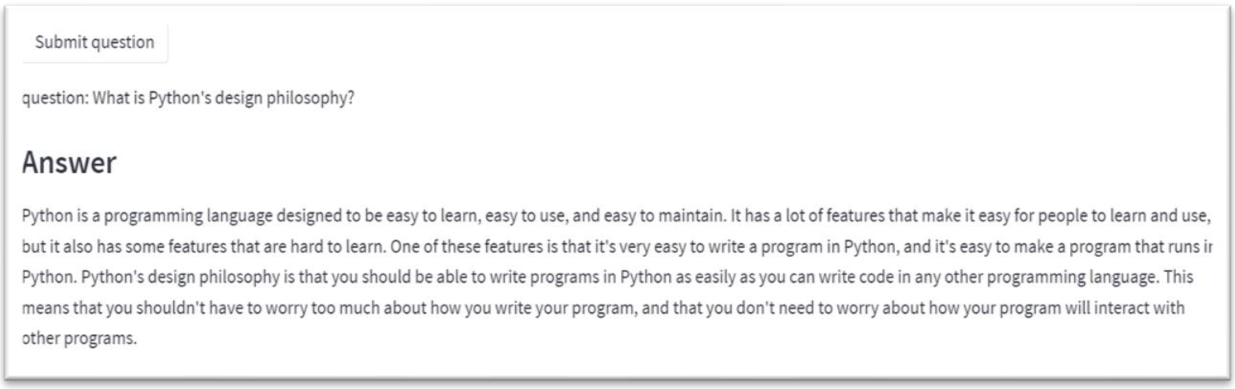


question: What does gravity give weight to on Earth?

Answer

Gravity doesn't give weight to anything on Earth. It gives weight to the Earth's rotation, and the Moon's gravity causes the tides on the surface of the Earth.

Figure 31: Submitting A generated question.



question: What is Python's design philosophy?

Answer

Python is a programming language designed to be easy to learn, easy to use, and easy to maintain. It has a lot of features that make it easy for people to learn and use, but it also has some features that are hard to learn. One of these features is that it's very easy to write a program in Python, and it's easy to make a program that runs ir Python. Python's design philosophy is that you should be able to write programs in Python as easily as you can write code in any other programming language. This means that you shouldn't have to worry too much about how you write your program, and that you don't need to worry about how your program will interact with other programs.

Figure 32: submitting an input question.

4. Perspectives

Considering the time conducted to this project it was difficult to apply all the ideas that can make this work even better than its current version, and among the functionalities that we can add in order to improve the functionality of this app:

- Using multiple models in order to compare the performances based on methods like Blue and Rouge methods in order and chose the ones that offers the best performances in transforming the documents and the questions, and also generating the questions.
- Adding the possibility of checking the top three retrieved documents by the retriever as a kind of prove or a validation to the generated question.
- If our model make it to the production level, and after being used many times, our vector database will become rich of transformed documents about open domains, which means that the application will be able to answer questions without the obligation of uploading a pdf document.
- Instead of only accepting pdf document we can add the possibility of extracting text numerous other file formats (text files, word files, ppt files…).
- We can add the option of using only the document provided by the user to answer the question instead of retrieving documents from the vector data base.
- Making the execution much faster if it's possible.
- Making it possible to generate and answer questions from different languages instead of being able only to answer and generate question in English.

Conclusion

In conclusion, our project's results demonstrate the usefulness of this application for extracting information from small to medium-sized documents, particularly by providing the option of suggesting questions to the user. This drives us to continue working on the project and implement more tests and features.

Additionally, working on this project allowed us to discover useful Python tools such as Haystack and Transformers, which were critical to achieving our results. It also provided a valuable opportunity to enhance our NLP skills.

Finally, it was rewarding to see our work being utilized and helping friends and students, which further motivates us to not stop here and keep improving the project.

5 References

1. https://blog.paperspace.com/how-to-train-question-answering-machine-learning-models/#:~:text=Question%2DAnswering%20Models%20are%20machine,given%20options%2C%20and%20so%20on.
2. https://arxiv.org/pdf/1810.04805.pdf
3. https://arxiv.org/pdf/2110.15253.pdf
4. https://arxiv.org/pdf/2104.10810.pdf
5. https://arxiv.org/pdf/1706.03762.pdf
6. https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1174/reports/2761153.pdf
7. https://medium.com/towards-data-science/transformer-attention-is-all-you-need-1e455701fdd9
8. https://ai.facebook.com/blog/longform-qa/
9. https://towardsdatascience.com/transformers-141e32e69591
10. https://www.youtube.com/watch?v=7kLi8u2dJz0&list=WL&index=9
11. https://personal.utdallas.edu/~sanda/paper_nle03.pdf
12. https://towardsdatascience.com/day-1-2-attention-seq2seq-models-65df3f49e263#:~:text=What's%20a%20Seq2Seq%20Model%3F,outputs%20another%20sequence%20of%20items.&text=In%20the%20case%20of%20Neural,the%20translated%20series%20of%20words.
13. https://towardsdatascience.com/how-to-read-data-files-on-s3-from-amazon-sagemaker-f288850bfe8f
14. https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270#:~:text=Training%20the%20language%20model%20in,10%25%20use%20the%20original%20word.
15. https://towardsdatascience.com/natural-language-processing-count-vectorization-with-scikit-learn-e7804269bb5e
16. https://www.guru99.com/seq2seq-model.html
17. http://jalammar.github.io/illustrated-transformer/
18. https://towardsdatascience.com/self-attention-5b95ea164f61#2beb
19. https://commons.wikimedia.org/wiki/File:Streamlit-logo-primary-colormark-darktext.png
20. https://towardsdatascience.com/word2vec-research-paper-explained-205cb7eecc30(w2v)
21. https://www.techtarget.com/searchenterpriseai/definition/language-modeling#:~:text=Exponential.,feature%20functions%20and%20n%2Dgrams.
22. researchgate.net/figure/A-visual-representation-of-a-single-block-in-a-recurrent-neural-network-RNN-Taken-from_fig2_336607800
23. https://www.researchgate.net/figure/Long-Short-Term-Memory-LSTM-cell-Fundamental-components-of-an-LSTM-cell-are-a-forget_fig1_329619613
24. https://inside-machinelearning.com/en/encoder-decoder-what-and-why-simple-explanation/
25. https://towardsdatascience.com/understanding-encoder-decoder-sequence-to-sequence-model-679e04af4346
26. https://arxiv.org/pdf/1909.11942.pdf