

CUSTOM CHATBOT USING LLAMA_INDEX AND OPENAI API

Anne-Sophia LIM

Supervisor: Dr Edwin FOO

May - July 2023



Acknowledgement

I am deeply grateful to my supervisor, Dr Edwin FOO whom I would like to thank for his invaluable guidance and support throughout my internship at NYP as he was always available to answer my questions and provide valuable feedback.

I would like to express my gratitude towards Mr. Patrick D'CRUZ and Mrs. Sabrina CLEMENT, for their help from the beginning of our integration in Singapore as they were the ones who coordinate the whole process to be as smooth as possible.



Table of Contents

Acknowledgement.....	2
Table of Contents.....	3
Introduction.....	4
Some definitions.....	5
• Definition of NLP.....	5
• Definition of LLM.....	5
• What is OpenAI?.....	5
• Different types of Openai endpoints.....	6
• The problem with the number of tokens.....	7
My chatbot code.....	8
• What is Llama_index?.....	8
• What are embeddings?.....	9
• My code.....	9
My data.....	12
• JSON files.....	12
◦ First draft.....	12
◦ The JSON files.....	14
• TEXT file.....	16
• XLSX files.....	16
Fine-tuning.....	18
Speech-to-text and text-to-speech.....	19
• Speech-to-text.....	19
• Text-to-speech.....	20
Graphical user interface with Streamlit.....	21
Some tests.....	23
Conclusion.....	25
Reference.....	26

Introduction

According to the website Statista.com, the volume of data available in 2020 was about 64 zettabytes, which means 64 trillion gigabytes and this number should triple by 2025. This incredible amount of information is so much to manage and sort that people can struggle to look for information efficiently. [1] That is why more and more automated help service agents are created on websites and that eventually led to the creation of chatbots like the very well-known ChatGPT. But the problem is that the latest data trained on are from September 2021. One solution to have the chatbot have access to the newest data is to train our chatbot but to optimize computational resources, minimize financial costs, and reduce training time, it should be trained using targeted data.

This project is about a custom chatbot using the OpenAI API which data comes from the Nanyang Polytechnic website. It can answer questions about two specific courses: Robotics and Mechatronics and Engineering with Business. This should have been implemented into a robot but I only worked on the Python code.

Some definitions

My chatbot was first created using the OpenAI API and OpenAI's Davinci model (which is based on the GPT-3 model). The Davinci model (and its other versions) is commonly used in tasks involving natural language processing. It has been trained to handle tasks that required an in-depth knowledge of the subject, such as realizing a summary of texts.

- Definition of NLP

First of all, we should define what NLP is. It stands for 'Natural Language Processing' and is a branch of computer science and in particular, the branch of Artificial Intelligence. NLP encompasses a broad range of tasks like text classification, sentiment analysis, machine translation, question answering, and more, using machine learning and deep learning models to deal with data. It can be encountered in digital assistants, speech-to-text dictation software or customer service chatbots. [2] [3]

- Definition of LLM

A Large Language Model is a trained deep learning model that can process and understand a large amount of text and generate a response in a human-like manner. They learnt from a massive number of data during pre-training with thousands of parameters. They are built as a transformer-based architecture, which is a neural network architecture. LLMs have the advantage to be very adaptive when it comes to learning new information and can be used through different applications. However, training LLMs can be expensive as powerful hardware should be used to be more efficient and there is a more or less significant risk of bias depending on the data that has been trained on. Examples of large language models include OpenAI's GPT models, or BERT (developed by Google). [4]

We can now talk about OpenAI, which plays a major role in the world of Artificial Intelligence, contributing significantly to its advancements. Its development has influenced various AI applications, such as natural language processing, computer vision, and reinforcement learning, shaping cutting-edge research in the AI community.

- What is OpenAI?

OpenAI is a private laboratory founded in 2015 in San Francisco, some of its founders are Ilya Sutskever (CSO), Greg Brockman (the actual chairman and president of openAI), Sam Altman (current CEO) and Elon Musk (before he leaves three years later due to potential conflict of interests because of his status as the CEO of Tesla). OpenAI aims to

develop safe artificial intelligence for the “benefit of all humanity”. They develop several LLMs through the past years, that can be accessed via its API.

In 2018 was released GPT-1, their first large language model. Two years later, GPT-3 was released, which is a model of pre-trained language model on a large dataset. The next year was the turn of Dall-E, which is a model that generates unique and creative images from textual descriptions. Then came ChatGPT in late 2022, a chatbot built on top of GPT-3.5 (this model is an improved version of GPT-3) and four months ago was released GPT-4. It can now understand visual input and push the limitation of the token limits to 8 192 tokens (GPT-3.5 limitation was up to 4 096 tokens).

Different models have been created based on GPT-3.5, they all have been fine-tuned to be more effective for chats. The one that I use for my chatbot is text-davinci-003. It generally produces a higher-quality response than the gpt-3.5-turbo model and can deliver longer outputs. [5]

The OpenAI API (application programming interface) allows developers to access and integrate the OpenAI’s LLMs and their natural language processing capabilities for different use cases through what are called endpoints. The ones that are the most fitted for our case are the chat/completion and the completion endpoints. [6]

● Different types of Openai endpoints

The completion or the chat completion both return answers from a given input. However, they need different parameters.

The chat completion endpoint needs as a parameter the model that will be used and a list of messages that will give an example of how it should respond, with the different roles. Here is an example:

```
{  
  "model": "gpt-3.5-turbo",  
  "messages": [{"role": "system", "content": "You are a helpful assistant."}, {"role": "user",  
"content": "Hello!"}]  
}
```

The message is generally a list of message role/content, that begins with the system, followed by an exchange between the user and the assistant. After giving the message, the model will give a response as the assistant he is learnt to be.

Other parameters can be set up, such as a function created by itself or implemented by the user that can be called when it is relevant to use it in the response, and a temperature that will create a more or less random or deterministic response, which I will explain below.

For the Completion endpoint, the main difference is that we enter a prompt instead of messages. The prompt can also contain the dataset we want it to learn. The model can then return one or more predicted completions. [7]

- The problem with the number of tokens

However, there is a limitation of tokens taken for each prompt. Depending on the model, for example, if we use the “text-davinci-003” or “chat-gpt-3.5” model, the limitation is 4097 tokens per request. But for the “chat-gpt-4”, the limitation reaches 8192 tokens per request (I did not use this model because to do so, I should have subscribed to ChatGPT Plus which costs 20\$ per month). [8]

The problem here is that the number of tokens for the file for Robotics & Mechatronics has each time exceeded the maximum token size allowed and I did not take into account the Engineering and Business file.

What I did first was extract the whole HTML code from the following website:

“<https://www.nyp.edu.sg/schools/seg/full-time-courses/robotics-and-mechatronics.html>”

The HTML file contained the code used to display the information on the website, so there were a lot of unnecessary lines. I removed them and save the new file as a text file. This one works but some of the data were not accurately processed (the chatbot cannot differentiate the different elective programmes of the third year) so I decided to create a JSON file containing only the useful information, thinking it would better structure the data. However, I came back to the original problem, which was the number of tokens exceeded as shown below.

```
InvalidRequestError: This model's maximum context length is 4097 tokens, however you requested 6504 tokens
```

I decided to try to use the Llama_index library, which is more flexible when it comes to data integration.

My chatbot code

I tried several tutorials to start my project. Some of them start from scratch, meaning that we had to train the chatbot to write the basic sentences, but as we cannot train the chatbot as much as ChatGPT did, it will never be as accurate and precise in its response so this was not the more optimal way to start. Others used the endpoints of Openai but to overcome the problem of the maximum number of tokens, I use Llama-index.

- What is Llama_index?

Llama_index is a data framework that helps us connect our LLM with external data. It provides tools such as data connectors that combine datasets from any source. They can be structured (from Excel or SQL for example), unstructured (like raw text files or PDFs) or semi-structured (these datasets come from APIs like Slack for example). It can also index our data. Indexing data allows a more efficient search over the dataset by creating a data structure optimized for retrieval. Even if the volume of data increases, it can manage the scalability and maintain a high search performance. Plus, it can be used to create embeddings to help with the relatedness of texts. Finally, It can supply a retrieval and query interface and can be integrated with other frameworks like LangChain or ChatGPT. With an input prompt over our data, it can return a knowledge-augmented response. [9] [10]

With Lama_index, I chose to use the “text-davinci-003” model. To do so, I had to put in an OpenAI API key. This key allows us to enter prompts and get back a response taking into account the rate limits but not the maximum number of tokens. This limit is used to regulate the number of times the server can be accessed within a precise period (contrary to the maximum number of tokens which limits, as said, the number of tokens per prompt). The reasons for limitation are variable, it ranges from protecting the server against misuse or abuse of the API to regulating the flow to give more fair access to other people using it without any slow service or interruption. They are measured in two ways: RPM (requests per minute) and TPM (tokens per minute). The table below shows the maximum amount of RPM and TPM allowed. We can see that the amount differs according to the different models used.

	TEXT & EMBEDDING	CHAT	EDIT	IMAGE	AUDIO
Free trial users	3 RPM 150,000 TPM	3 RPM 40,000 TPM	3 RPM 150,000 TPM	5 images / min	3 RPM
Pay-as-you-go users (first 48 hours)	60 RPM 250,000 TPM	60 RPM 60,000 TPM	20 RPM 150,000 TPM	50 images / min	50 RPM
Pay-as-you-go users (after 48 hours)	3,500 RPM 350,000 TPM	3,500 RPM 90,000 TPM	20 RPM 150,000 TPM	50 images / min	50 RPM

image: <https://platform.openai.com/docs/guides/rate-limits/overview>

Moreover, the number of tokens is not equivalent to the old models. In the below table we can see that for example, 1 TPM is equal to 25 TPM for the curie model.

TYPE	1 TPM EQUALS
davinci	1 token per minute
curie	25 tokens per minute
babbage	100 tokens per minute
ada	200 tokens per minute

- What are embeddings?

I started my code with a function called `createVectorIndex()` that is used for embeddings. I will first explain what embeddings are.

An embedding can be described as a numerical representation of some words or even sentences through vectors. It can transform “high-dimensional vectors into a lower-dimensional space”, to quote the Leewayhertz website, to capture the meaning of the words by placing similar ones near them.

To do so, the data inserted is divided into chunks, where embeddings are created for each chunk. The words used in the same context will have similar vectors. By creating a list of embeddings, they can work as a dictionary, which the model will learn from to obtain more information.

Embeddings can be used in other different cases, like text classification or sentiment analysis. [11] [12]

- My code

The first part of this function is creating a prompt helper with hyperparameters.

```
def createVectorIndex():  
    max_input = 4096  
    tokens = 256  
    max_chunk_overlap = 20  
    chunk_size = 600  
  
    prompt_helper = PromptHelper(max_input, tokens, max_chunk_overlap, chunk_size_limit=chunk_size)
```

The PromptHelper class helps us to split the dataset into smaller chunks according to the limitations we put.

The second part is the creation of the LLM, based on the OpenAI’s model “text-davinci-003”.

```
#define LLM
llmPredictor = LLMPredictor(llm=OpenAI(
    temperature=0.5,
    model_name="text-davinci-003",
    max_tokens=tokens
```

The LLMPredictor class is wrapper around an LLMChain from LangChain. It handles the conversion from the prompt input it has received to a string that can be understood by the LLM put inside. The hyperparameter temperature here controls the randomness and thus the creativity of the response of the LLM. The higher the temperature is, the more random the response will be.

For example, I ask the chatbot “What are the elective programmes of robotics?”, this is the response I got with a temperature of 0.2:

“The two elective programmes of robotics are Automation & Robotics Technology and Wafer Fabrication Technology. ”

For the temperature of 0.8:

“The two elective programmes of robotics are Automation & Robotics Technology and Aerospace Technology.”

We can see that the first response is correct whereas the second one is more random (and half false).

For our case, as we want an accurate answer, we can choose a temperature between 0.2 and 0.5.

The function createVectorIndex() then loads the documents taken from a folder thanks to the SimpleDirectoryReader() function, creates indexes from these documents (with the GPTVectorStoreIndex()’s from_documents() function) and sets the “index_id” (thanks to set_index_id()). We have to persist the indexes to allow the LLM to have access to them without having to create the indexes again. It is saved into a file named “vector_store.json” [13]

```
# load documents
documents = SimpleDirectoryReader('D:\ESIEE\VOYAGE SINGAP 2023\project_custom_chatbot_nyp\my_data').load_data()
vectorIndex = GPTVectorStoreIndex.from_documents(documents=documents, llm_predictors=llmPredictor, prompt_helper=prompt_helper)

# save index to the folder: "vectorIndex"
vectorIndex.set_index_id("vector_index")
vectorIndex.storage_context.persist('vectorIndex')
```

This function is saved separately in another Python file.

The second function of my code answerMe(vectorIndex) is used for saving the indexing to the storage with the storage_context.persist() function.

```
def answerMe(prompt):  
    storage_context = StorageContext.from_defaults(persist_dir='vectorIndex')  
    vIndex = load_index_from_storage(storage_context, index_id="vector_index")  
    query_engine = vIndex.as_query_engine()  
    response = query_engine.query(prompt)  
    return(response)
```

It allows the LLM to load and have access to it when creating the query engine with the `as_query_engine()` function. The user can interact with the chatbot through the input which will send the query to the query engine, which will then deliver the response.

My data

The data put inside my code contains different types of files, for different purposes. The model uses the JSON and text files to learn (once they are transformed into embeddings) and two XLSX files to help it to be more precise.

- JSON files

I got my data from these two websites:

- NYP Engineering with business:

<https://www.nyp.edu.sg/schools/seg/full-time-courses/engineering-with-business.html>

- NYP Robotics and Mechatronics:

<https://www.nyp.edu.sg/schools/seg/full-time-courses/robotics-and-mechatronics.html>

- First draft

I first simply scraped these whole websites (one JSON file for each website) with BeautifulSoup:

```
url = "https://www.nyp.edu.sg/schools/seg/full-time-courses/robotics-and-mechatronics.html"
response = requests.get(url)
soup = BeautifulSoup(response.content, "html.parser")
```

With this code, it sends a request to get the url using the requests library. It then creates a BeautifulSoup object by parsing the HTML content of the response. This allows us to navigate and extract data from the webpage using BeautifulSoup's methods.

The main problem is that (as said above) the files' size exceeded the maximum number of tokens allowed and while most of the file was correctly sorted, a small part was not. Especially the third year of Robotics and Mechatronics which has two elective programmes: Automation & Robotics Technology and Wafer Fabrication Technology: when the website is converted to a text file, each line was written one below the other and with that, the model cannot properly handle the information.

What the website looks like when we pull down the menu of year 3:

Elective Programmes

(Choose one specialisation)

Automation & Robotics Technology

Core Modules

- [Automation Systems Design](#)
- [Communication & Personal Branding](#)
- [Motion Control & Drives](#)
- [Semestral Project](#)
- [General Studies](#)

Elective Modules (Choose two)

- [Communication & Networking](#)
- [Intelligent Systems](#)
- [Mechanisms Design & Simulation](#)
- [Systems & Control](#)
- [Wafer Fabrication Processes](#)

Wafer Fabrication Technology

Core Modules

What it looks like once as a text file:

```
Year 3
Elective Programmes
(Choose one specialisation)
Automation & Robotics Technology
Core Modules
Automation Systems Design
Communication & Personal Branding
Motion Control & Drives
Semestral Project
General Studies
Automation Systems Design [60 hours]
This module aims to provide essential concepts and principles t
```

When I tested my code with a part of the file (only year 3), with this question:

```
ans = mychatbot('with the given information only, what are the different specialisation for year 3?')
```

I get this response: 'The specialisations for Year 3 are Automation & Robotics Technology, Communication & Networking, Intelligent Systems, Mechanisms Design & Simulation, Systems & Control, and Wafer Fabrication Processes.'

But when I try:

```
ans = mychatbot('what are the different specialisation for year 3?')
```

This is what I get: 'Year 3 offers students four specialisation options: Automation & Robotics Technology, Wafer Fabrication Technology, Biomedical Engineering, and Mechatronics & Automation Technology. Each specialisation offers its own sets of core and elective modules.'

We can see here that the model is a bit more accurate (still false because two of the specialisation have been removed since then) with the second question, probably because it has checked its information instead of the file I provided.

I wanted to see if I could get better results for the first question with a proper JSON file, where the information would be sorted.

○ The JSON files

I will only talk about the JSON file for Robotics and Mechatronics as it was more arduous than the one for Engineering with Business, but they both have the same pattern overall.

Most of the code for Robotics and Mechatronics follows the same scheme, especially for year 1 and year 2 but year 3 was more challenging as it has two elective programmes with core modules and elective modules themselves. I first did not know how to sort the JSON file or what should be nested first, if I should separate by core and elective modules or by years... After some tests, it appears that organizing by years was more efficient.

I created a function called `creation_df_years(soup, years)` that returns a dataframe for each year containing the name of the module, the number of hours required and its description.

The second function called `split_title_description(df_year)` splits the dataframe into three parts: title, hours and description which are converted to lists.

I then adjusted these lists by removing or adding some lines following the modules because some lines should be concatenated while some parts should be split.

Here is what I did for year 1 (the year 2 behaves the same way):

```
# Year 1
title_tab_1 = split_title_description(df_year_1.iloc[:,0])[0]
hours_tab_1 = split_title_description(df_year_1.iloc[:,0])[1]
description_tab_1 = split_title_description(df_year_1.iloc[:,0])[2]
description_tab_1 = concat_lines(5,6,description_tab_1) # append the note that was below the description
description_tab_1 = concat_lines(len(description_tab_1) - 2, len(description_tab_1), description_tab_1)
```

I split and saved the dataframe of the year into three different variables. I had to concatenate two parts of the table `description_tab` (because some lines have been separated while they should not).

When I print `df_year_3`, this is what I get:

We can see that the two elective programmes are not distinguishable. To separate automatically the two parts and their core and elective modules, I looked for the indexes of the lines containing 'core modules' ([2, 26]) and 'elective modules' ([14, 38]).

The end of the first table 'core modules' is when begins the table 'elective modules'

```
Year 3
0 Elective Programmes\n(Choose one specialisation)
1 Automation & Robotics Technology
2 Core Modules
3 Automation Systems Design [60 hours]
4 This module aims to provide essential concepts...
5 Communication & Personal Branding [30 hours]
6 In today's competitive environment, a strong p...
7 Motion Control & Drives [60 hours]
8 This module equips learners with practical kno...
9 Semestral Project [60 hours]
10 This module develops learners' abilities in ap...
11 General Studies
12 To provide you an all-rounded education, NYP o...
13 To learn more about the GSMS offered, click here
14 Elective Modules (Choose two)
15 Communication & Networking [60 hours]
16 This module acquaints learners with the fundam...
17 Intelligent Systems [60 hours]
18 This module provides students with essential k...
19 Mechanisms Design & Simulation [60 hours]
20 This module aims to equip learners with knowle...
21 Systems & Control [60 hours]
22 This module provides learners with the fundame...
23 Wafer Fabrication Processes [60 hours]
24 This module aims to equip learners with the fu...
25 Wafer Fabrication Technology
```

(index of the elective module - 1), I applied this principle to the different tables.

```
index = 0
indexes_core_modules = df_year_3.index[df_year_3['Year 3'].str.contains('Core Modules')] # get the indexes of df_year_3 that
indexes_elective_modules = df_year_3.index[df_year_3['Year 3'].str.contains(r'Elective Modules.*', regex=True, case=False)]

if indexes_core_modules[index] != indexes_core_modules[-1]:
    core_modules_EP_1 = df_year_3[indexes_core_modules[index]:indexes_elective_modules[index]-1]
    elective_modules_EP_1 = df_year_3[indexes_elective_modules[index]:indexes_core_modules[index+1]-1]
    name_EP_1 = df_year_3.iloc[indexes_core_modules[index]-1]
    index+=1
core_modules_EP_2 = df_year_3[indexes_core_modules[index]:indexes_elective_modules[index]-1]
elective_modules_EP_2 = df_year_3[indexes_elective_modules[index]:]
name_EP_2 = df_year_3.iloc[indexes_core_modules[index]-1]
```

As the variables here are still pandas objects, I had to convert the names into a string to retrieve them.

```
# get the names of the elective programmes
name_EP_1 = ''.join(name_EP_1)
name_EP_2 = ''.join(name_EP_2)
```

Finally, I separated each module's tables into three new variables (title, hours, description).

```
# elective programme 1 : automation & robotics technology core module and elective module
title_tab_3_EP_1_CM = split_title_description(core_modules_EP_1.iloc[1:,0])[0]
hours_tab_3_EP_1_CM = split_title_description(core_modules_EP_1.iloc[1:,0])[1]
description_tab_3_EP_1_CM = split_title_description(core_modules_EP_1.iloc[1:,0])[2]

title_tab_3_EP_1_EM = split_title_description(elective_modules_EP_1.iloc[1:,0])[0]
hours_tab_3_EP_1_EM = split_title_description(elective_modules_EP_1.iloc[1:,0])[1]
description_tab_3_EP_1_EM = split_title_description(elective_modules_EP_1.iloc[1:,0])[2]

# elective programme 2 : Wafer Fabrication Technology core module and elective module
title_tab_3_EP_2_CM = split_title_description(core_modules_EP_2.iloc[1:,0])[0]
hours_tab_3_EP_2_CM = split_title_description(core_modules_EP_2.iloc[1:,0])[1]
description_tab_3_EP_2_CM = split_title_description(core_modules_EP_2.iloc[1:,0])[2]

title_tab_3_EP_2_EM = split_title_description(elective_modules_EP_2.iloc[1:,0])[0]
hours_tab_3_EP_2_EM = split_title_description(elective_modules_EP_2.iloc[1:,0])[1]
description_tab_3_EP_2_EM = split_title_description(elective_modules_EP_2.iloc[1:,0])[2]
```

I extracted other relevant information from the website such as the contact information of the course coordinators and manager, or the description of the career prospects.

I insert all the information into my JSON object, here is an extract:

The JSON object is then saved into a new JSON file named 'R&M.json'.

```
data = {
    "fullTimeCourses": [
        {
            "name": full_time_course_name,
            "about the course": info,
            "years": []
        }
    ],
    "career Prospects": description_career_prospects,
    "further Studies": description_further_studies,
    "contact Us": {
        "course Manager": {
            "name": contact[1],
            "telephone Number": contact[2],
            "email": contact[3]
        },
        "course Coordinator": [
            {
                "name": coord[1],
                "telephone Number": coord[2],
                "email": coord[3]
            },
            {
                "name": coord[5],
                "telephone Number": coord[6],
                "email": coord[7]
            }
        ]
    }
}
```

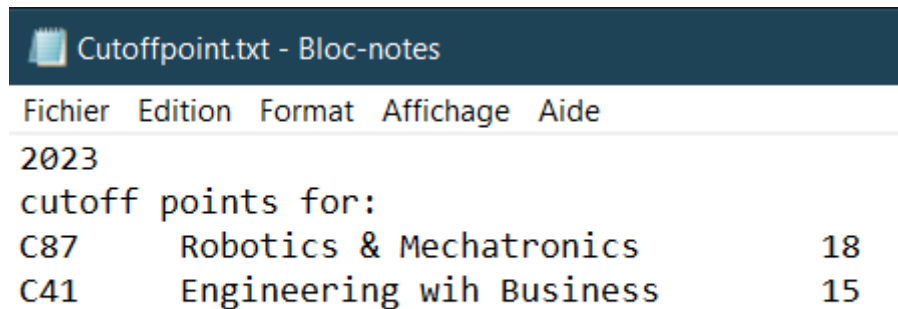


```
with open('R&M.json', 'w') as json_file:
    json.dump(data, json_file, indent=1)
```

I did the same for Engineering with Business, and saved it into a JSON file named "EWB.json".

- TEXT file

This text file given by my supervisor added more information about the courses that were not provided on the websites.



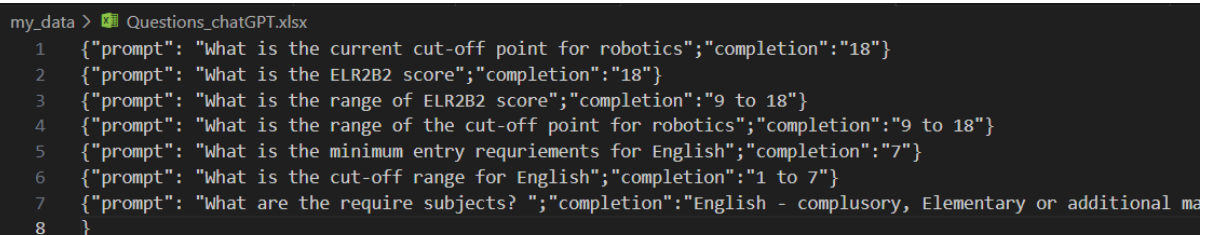
2023		
cutoff points for:		
C87	Robotics & Mechatronics	18
C41	Engineering with Business	15

The text file

I then fine-tuned my model as the responses were sometimes a bit random. I will explain below the principle of fine-tuning.

- XLSX files

This file was also given by my supervisor, it added some new information for the model to learn.



```
my_data > Questions_chatGPT.xlsx
1  {"prompt": "What is the current cut-off point for robotics";"completion":"18"}
2  {"prompt": "What is the ELR2B2 score";"completion":"18"}
3  {"prompt": "What is the range of ELR2B2 score";"completion":"9 to 18"}
4  {"prompt": "What is the range of the cut-off point for robotics";"completion":"9 to 18"}
5  {"prompt": "What is the minimum entry requirements for English";"completion":"7"}
6  {"prompt": "What is the cut-off range for English";"completion":"1 to 7"}
7  {"prompt": "What are the require subjects? ";"completion":"English - compulsory, Elementary or additional ma
8  }
```

First XLSX file

The last file was created for the chatbot to be more precise in its response.

```

my_data > QA1.xlsx
1  {"prompt": "What is the main focus of the ""Diploma in Robotics & Mechatronics"" course?";"completion": "The course focuses on combining me
2  "}
3  {"prompt": "What specialization areas are offered in the ""Diploma in Robotics & Mechatronics"" course?";"completion": "The course offers spe
4  "}
5  {"prompt": "What topics are covered in the ""Algebra"" module? ";"completion": "The ""Algebra"" module covers mathematical functions like po
6  "}
7  {"prompt": "What skills will I develop in the ""Effective Communication Skills"" module? ";"completion": "In this module, you'll learn to co
8  "}
9  {"prompt": "What knowledge will I gain from the ""Electrical Principles"" module? ";"completion": "The ""Electrical Principles"" module cover
10 }
11 {"prompt": "What does the ""Engineering Drawing & Modelling"" module entail? ";"completion": "This module covers the knowledge and skills to
12 "}
13 {"prompt": "What can I expect to learn from the ""Fundamentals of Innovation and Enterprise"" module? ";"completion": "In this module, you'll
14 "}

```

Extract of the second XLSX file

Fine-tuning

Fine-tuning is a process that will enhance the data of the LLM without having it entirely be trained again because training an LLM can be time-consuming, especially if it has thousands of gigabits of data to learn. I realized I did not properly fine-tune my model because fine-tuning the OpenAI's model requires its commands, and by the time I found some useful articles about that I was short on time so I chose to focus on finishing the interface of my chatbot. However, the XLSX files have been written in the right form to fine-tune the model later, with this scheme:

“{“prompt”: “question”; “completion”: “answer”}” (cf. see above for the XLSX files)

It is based on a pre-trained model, where we retrieve the weights of a trained neural network that has been obtained from the previous training instead of putting random weights. This is especially useful when we do not have a large amount of data and a model that can already be used. To get these weights, we have to freeze the layers of the neural network, but not entirely. We can change the number of layers to freeze to try to have a more accurate model.

Here is an example of two neural networks, the left one has been trained without freezing the layers while the right one has some layers that have been frozen, so the weights are more adjusted. [14] [15]

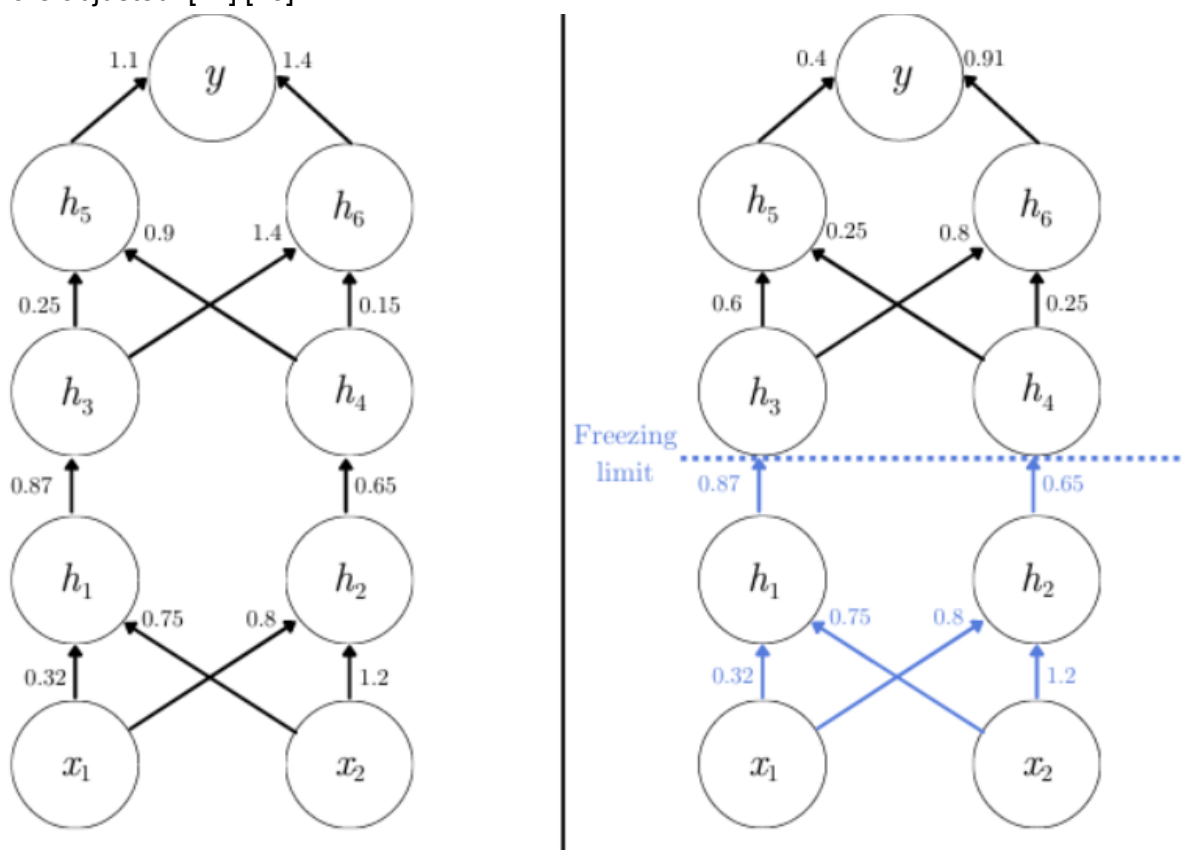


image: <https://www.baeldung.com/cs/fine-tuning-nn>

To fine-tune the OpenAI's models, this website shows us how to do it:
<https://platform.openai.com/docs/guides/fine-tuning/preparing-your-dataset>

Speech-to-text and text-to-speech

This part of the code was to enhance the chatbot in order to recreate a more human-like chatbot.

- Speech-to-text

The speech-to-text code is used when the user wants to ask a question orally.

I used the SpeechRecognition library.

```
import speech_recognition as sr
```

This is a Python library that can integrate speech recognition functionality such as handling audio inputs and saving them, sending them to an API or retrieving and also translating the recognized text.

I created a function called “speech-to-text(timeout=10)” which will recognize the speech input within 10 seconds after running the function (the duration of the parameter can be adapted).

I initialized an instance of the Recognizer class from the SpeechRecognition library, which handles speech recognition operations.

```
# Initialize recognizer class (for recognizing the speech)
recognizer = sr.Recognizer()
```

```
for i in range(3): # will retry 3 times
    # Record Audio
    with sr.Microphone() as source:
        print("Say something!")
        # Optionally, can add this line to adjust for ambient noise
        # recognizer.adjust_for_ambient_noise(source) #listen for 1 seco
        # Listening to the microphone
        audio = recognizer.listen(source, timeout=timeout)
        try:
            # Store the result in a variable
            text = recognizer.recognize_google(audio, language='en-US')
            return text # if speech recognition was successful, return t
        except sr.UnknownValueError:
            print("Sorry, I did not understand, can you repeat please?")
        except sr.RequestError as e:
            print("Request Failed; {0}".format(e))

print("Sorry, I still did not understand what you said.")
return None
```

I then set up a loop that will try three times to recognize the speech, in case it does not understand what the user said. The microphone of the computer is used thanks to the Microphone class from the library to detect and record the audio, which is passed to the “recognize_google()” function. This function is used to understand the speech and convert it into a text stored in a variable “text”.

The function is created to properly handle errors, otherwise, it would crash each time there is an error. [16]

- Text-to-speech

This part is used to convert text input to spoken words. The library used here is pyttsx3.

```
import pyttsx3
```

This library allows us to customize the voice output, such as the language of the voice or its speed.

```
def text_to_speech(mytext):  
    # The text that you want to convert to audio  
    engine = pyttsx3.init()  
    engine.setProperty('voice', 'HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Speech\Voices\Tokens\TTS_MS_EN-US_ZIRA_11.0')  
    engine.setProperty("rate", 187) # Integer speech rate in words per minute. Defaults to 200 word per minute.  
    engine.say(mytext)  
  
    return engine.runAndWait()
```

I created a function called `text_to_speech(mytext)`, the parameter is the string that will be spoken. The engine that is initialized is responsible for generating the speech audio.

I then customize some parameters. As my laptop language is in French, the default language used there is French, which means that even if the sentence is in English, the voice would say it with a French accent, so I had to manually set the voice to an English voice.

The speech rate is also changed and is set to slightly slow it down.

The “`engine.say(mytext)`” line tells the engine to convert the specified text into speech. Finally, “`engine.runAndWait()`” is called to run the text-to-speech engine and wait for the speech to complete. It ensures that the program doesn't proceed until the speech is finished.

[17]

Graphical user interface with Streamlit

Streamlit is an open-source Python library that allows developers to create web applications for data science and machine learning projects easily. It is designed to help developers to convert data scripts into interactive web applications that can be accessed through a web browser. Streamlit's user-friendly nature makes it an easily accessible and approachable framework for beginners.

I created a new Python file called app.py that includes almost all the code from the previous file but to avoid problems between the front-end and the back-end, I change some parts of the code, for example, the three-times trial from the speech-to-text function.

However, I added a voice trigger function that will detect a specific phrase before processing the speech, so the speech detection can then be run in the background non-stop.

Here is my code:

```
# Voice trigger function
def listen_for_trigger():
    recognizer = sr.Recognizer()
    mic = sr.Microphone()
    trigger_phrase=["hey assistant", "thank you"]
    print("Listening for trigger...")
    with mic as source:
        while True:
            audio = recognizer.listen(source)
            try:
                trigger = recognizer.recognize_google(audio).lower()
                if trigger_phrase[0] in trigger:
                    print("Trigger detected.")
                    return trigger_phrase[0]
                if trigger_phrase[1] in trigger:
                    print("You're welcome, bye!")
                    return trigger_phrase[1]
            except sr.UnknownValueError:
                pass
            except sr.RequestError:
                print("Sorry, speech recognition service is currently unavailable.")
                break
```

There are two triggers "Hey assistant" and "Thank you". When the recognizer will identify one of them, the latter will be returned.

Here is the main function:

```

def main():
    trigger_phrase = ["hey assistant", "thank you"]
    st.title("Voice Assistant with Streamlit")
    start_listening = st.button("Start Listening")
    stop_listening = st.button("Stop Listening")

    def response():
        st.write(f"Question: {prompt}")
        response = answerMe(prompt)
        st.write(f"Response: {response}")
        text_to_speech(response)
        return

    if start_listening:
        st.write("Listening for trigger...")
        while True:
            if listen_for_trigger() == trigger_phrase[0]:
                text_to_speech("How can I help you?")
                st.write("Trigger detected. What is your question?")
                prompt = speech_to_text(timeout=30)
                response()
            if listen_for_trigger() == trigger_phrase[1]:
                st.write("You're welcome!")
                text_to_speech("You're welcome!")
                break
            if stop_listening:
                break

```

The website is built so there are only the title and the two buttons “start listening” and “stop listening”. However, there is one problem with these buttons, when I click on the button “stop listening”, it actually does not stop the back end, only the front end part, so the code keeps running in the background and causes errors when I click on the button “start listening” again. This is a problem that I still did not solve due to a lack of time.

This is a screenshot of the webpage of my chatbot. When I tested the chatbot, it works in general but have some problems. Sometimes, the chatbot stops us in the middle of our sentence, but I decided to leave it like this because if I have allowed more time for us to speak, for example, 60 seconds, and our question only lasts 15 seconds, we would have to wait a long time before getting the response and it could have interfered with other sounds.

Voice Assistant with Streamlit

Start Listening

Stop Listening

Listening for trigger...

Trigger detected. What is your question?

Question: what are the required subjects

Response: The required subjects are AI Applications, Analogue & Digital Electronics, Automatic Control, Device Interfacing & Programming, Differential Equations & Series, Mechanical Design, Microcontroller Applications, Probability & Statistics, Quality Assurance, Robotic Systems & Peripherals, Semestral Projects, Semestral Project 3, Semestral Project 4, and General Studies.

Trigger detected. What is your question?

Question: range for English

Response: 1 to 7

Some tests

I tested my chatbot with a set of 20 questions that I wrote into a text file named "questions_chatbot.txt". I saved the response I got into a text file and look into it.

```
What is the cut off point for engineering with business?
Who can I contact to have more information about engineering with business course?
what are the career prospects after graduating from engineering with business?
what can you tell me about engineering with business?
what are the core modules for year 3 in engineering with business?
How will the "Engineering Exploration Project" module benefit me as a learner?
Could you provide more details about the "Materials Technology" module?
What are the key topics covered in the "Workplace Digital Skills" module?
Can you explain what is the course "business management" about?
What are the elective modules of "Automation & Robotics Technology" in robotics and mechatronics?
```

Extract from my file "questions_chatbot.txt"

Here is the answer I got in the "output_responses.txt" file:

```

Query: What is the cut off point for engineering with business?
Response:
The cut off point for engineering with business is 15.
Query: Who can I contact to have more information about engineering with business course?
Response:
You can contact the Course Manager, Dr. Ang Wei Sin, for more information about the Diploma in Engineering with Business cou
Query: what are the career prospects after graduating from engineering with business?
Response:
Career prospects after graduating from engineering with business include business planning and development, project planning
Query: what can you tell me about engineering with business?
Response:
Engineering with Business is a diploma course offered by NYP that combines engineering and business concepts. It covers a wi
+++Query: what are the core modules for year 3 in engineering with business?
Response:
The core modules for Year 3 in Engineering with Business are: Automation Systems Design, Communication & Personal Branding,

```

Extract from the "output_responses.txt" file.

For each question, when the answer differ from what expected, I wrote "+" in front of the query. The more "+" it has, the more the answer is wrong.

For example, the last answer is completely false as it does not match the Engineering with Business modules but the Robotics and Mechatronics modules.

On a total of 20 questions, 16 are correct. As for now, I have a ratio of 80% of correct answers, but this result should be tested again with more questions.

Conclusion

To conclude, I learnt a lot from this project about how a chatbot works and I have a better understanding of large language models in general. This project can be improved in many ways, and its final purpose is to be integrated into the robot temi, an assistant robot that will help and answer questions from students of NYP.

By doing this project, I realise how much we use AI in every field, and how fast it is evolving, as some of the tutorials I saw were already deprecated when they were released only a few months ago, and I am sure that my code will also be in a few months.

It was a very enriching experience and confirms my choice of working in this field when I will get my diploma.

PS: my work is available here: https://github.com/Chocolalie/custom_chatbot_NYP_robotics

Reference

- [1] Taylor P., (2021) "Amount of data created, consumed, and stored 2010-2020, with forecasts to 2025.", Statista, June. Available at: <https://www.statista.com/statistics/871513/worldwide-data-created/> (Accessed: June 19, 2023).
- [2] "What is Natural Language Processing?", IBM, Available at: <https://www.ibm.com/topics/natural-language-processing#:~:text=the%20next%20step-What%20is%20natural%20language%20processing%3F,same%20way%20human%20beings%20can.> (Accessed: June 23, 2023).
- [3] (2023) "Computational Linguistics.", Coursera, Available at: <https://www.coursera.org/articles/computational-linguistics> (Accessed: July 2, 2023).
- [4] Kerner S. M. (2023), "Large Language Model (LLM).", TechTarget, Available at: <https://www.techtarget.com/whatis/definition/large-language-model-LLM> (Accessed: July 9, 2023).
- [5] Monge, J. C., (2022) "New GPT-3 Model Text-DaVinci-003 Is Awesome", Medium, December 8, Available at: <https://medium.com/technology-hits/new-gpt-3-model-text-davinci-003-is-awesome-ada11ef660a9> (Accessed: July 15, 2023).
- [6] Hashemi-Pour C., (2023) "OpenAI", TechTarget, March, Available at: <https://www.techtarget.com/searchenterpriseai/definition/OpenAI> (Accessed: July 23, 2023).
- [7] "OpenAI API Reference - Completions Create", OpenAI, Available at: <https://platform.openai.com/docs/api-reference/completions/create> (Accessed: July 23, 2023).
- [8] "GPT-3.5 Models Documentation", OpenAI, Available at: <https://platform.openai.com/docs/models/gpt-3-5> (Accessed: July 28, 2023).
- [9] "LLama Index", LLama Index, Available at: <https://www.llamaindex.ai/> (Accessed: July 29, 2023).
- [10] "GPT-Index Documentation", GPT-Index, Available at: <https://gpt-index.readthedocs.io/en/stable/> (Accessed: August 2, 2023).
- [11] Takyar A. "What is Embedding?", LeewayHertz, Available at: <https://www.leewayhertz.com/what-is-embedding/> (Accessed: August 7, 2023).
- [12] Jiang J., (2021) "What is Embedding and What Can You Do With It?", Towards Data Science, May 5, Available at:

<https://towardsdatascience.com/what-is-embedding-and-what-can-you-do-with-it-61ba7c05efd8> (Accessed: August 7, 2023).

[13] Lee W.-M., (2023) "Connecting ChatGPT with Your Own Data Using LLaIndex", Medium, May 13, Available at: <https://levelup.gitconnected.com/connecting-chatgpt-with-your-own-data-using-llamaindex-663844c06653> (Accessed: August 8, 2023).

[14] Barreto S., (2023) "Fine-Tuning Neural Networks", Baeldung, June 16, Available at: <https://www.baeldung.com/cs/fine-tuning-nn> (Accessed: August 8, 2023).

[15] Grinberg G., (2023) "OpenAI API Fine-Tuned Models vs. Chat Completion: A Case Study", Better Programming, June 1, Available at: <https://betterprogramming.pub/openai-api-fine-tuned-models-vs-chat-completion-a-case-study-e3774fadc8c7> (Accessed: August 8, 2023).

[16] (2021) "How to Build a Digital Virtual Assistant in Python", ActiveState, April 19, Available at: <https://www.activestate.com/blog/how-to-build-a-digital-virtual-assistant-in-python/> (Accessed: August 9, 2023).

[17] Niranjana Raguraman N., (2019) "Get Started with Speech-to-Text and Text-to-Speech in Python", Medium, March 7, Available at: <https://medium.com/@NiranjanaRaguraman/get-started-with-speech-to-text-and-text-to-speech-in-python-61ea4412696f> (Accessed: August 9, 2023).