My Personal AI Project

For context

This document is going to be expanded as I learn and build my project. I have the whole thing planned, but first I have to learn how to make everything work, and for that I need to understand everything as a whole at least on a somewhat basic level. For this, I will be using the Internet and **ChatGPT** to learn what I have to. This isn't the first day that I do research and testing on this, but it was today, specifically during my education in high-school, where a teacher I was talking with about my project advised me to keep everything I do on record. It's not a concept I'm unfamiliar with, because I have already done some research and projects that I documented, but I didn't think of doing it for this exact project until now. I must clarify that, if I refer to this project in different ways as I advance forward, please take into consideration that this starts as a personal project because I can't stand not understanding some things. As of 10th of October, 2024, the title of this document is "My Own AI Project".

All that explained, I will be putting comments inside of this very same document with the comments feature that Google Docs offers so it's easier for everyone, myself included, to keep track of what I'm doing on a timeline.

I have already understood how a language model understands text, how to train one, and a bit more. Now, I'm working on a program that will generate conversations for me in order to make the whole process of gathering a good quality database for training.

Automated Database Maker

I think that I could call this program ADM or maybe ADMa, because it sounds better and closer to an actual name, and I like giving human-like names to things I do. For now, I already asked ChatGPT to give me a bunch of nouns that make reference to physical places, like mansion, castle, pool… This was then formatted in JSON format so I can load it in my program instead of having to write this database in the very code. The result will be on its own page so it's easier to look at.

I learned that in JSON format, you have to, first, state objects, which would be like a variable in Python: a name to which you can assign a value in order to have it easier when working with said values. In this case, the JSON format I used states that the object called "noun" can be any of those values, or, for humans, words. Because it's text, it needs to be put in between these symbols (", "). Each time you finish writing a value, whether it is text or numbers, you put a comma (,) before putting in the next one. However, it is an error to put a comma after you have presented the last value, the last object, etc. Comma is only for separation, and you will run into errors for not remembering it. Because my plan is to test if I'm able to make a program to make a database faster (instead of having to write or copy and paste all the text I need and then format it), I believe this set of words is enough.

Now, although I technically could do a new object inside of this same JSON file creating a new array with the brackets symbols. That would look cleaner in my files, but I like the idea of having different databases more. I think, outside of personal taste, that different databases will be better for when I run into troubles with anything that's inside of them. The next thing I need to do is a database for adjectives that can be used with the nouns I have, like cozy, scary, big… I will then format it the same way, and load it into my program.

After getting confirmation from ChatGPT that this format is correct, I save it as "adjectives_database" the same way I did with "nouns_database".

Now, after importing these two, I will need to make a function to select a random word both in nouns and adjectives and then put them together. For that, first I will show, because I didn't show it before, how I load the database.

```
#This will load the databases I have

with open('nouns_database.json', 'r') as file: #Tells the program to
not only open the file described to be opened and read, but also be
closed after and be named "file" for shorten
    nouns_database = json.load(file) #This is for creating a variable
that refers to loading a json file in specific, in this case our
database named "file"

with open('adjectives_database.json', 'r') as file2: #same thing as
before, but with file2 because file is already in use
    adjectives_database = json.load(file2)
```

I was going to do it after, but because I would have to come back to it later, I will add one more database to it. This database will be "actions_database". It will contain actions that can be put after "are" in a sentence and make sense. You'll see why in a second

And I load this database too.

```
#This will load the databases I have

with open('nouns_database.json', 'r') as file:                    #Tells
the program to not only open the file described to be opened and read, but
also be closed after and be named "file" for shorten
```

```
    nouns_database = json.load(file)                                    #This
is for creating a variable that refers to loading a json file in specific, in
this case our database named "file"

with open('adjectives_database.json', 'r') as file2:                    #same
thing as before, but with file2 because file is already in use
    adjectives_database = json.load(file2)

with open('actions_database.json', 'r') as file3:
    actions_database = json.load(file3)
```

This is how all 3 databases being loaded looks like.

And now, with the databases set, I can start working on the context generator. The context generator will use one word from the adjectives, one word from the nouns and finally one action from the actions. This means that I will be creating a sentence that will look like this:

“In a cozy living room, where char1 and char2 are painting oranges.”

I will divide the context in two things: setting (adjective + noun) and action. To make this happen, the code looks like this:

```
def generate_random_setting():

    noun = random.choice(nouns_database["noun"])                        #Get
a random noun from the database
    adjective = random.choice(adjectives_database["adjective"])         #Get
a random adjective from the database

                    setting     =     f"in     a     {adjective}     {noun}"
#Making a variable named setting that will be an adjective and a noun
randomly selected. I use f string function to be able to put text in it
    return setting                                                      #This
gives back the result, like "in a cozy cafe"
```

And now I need to get a random action:

```
def generate_random_action():

    action = random.choice(actions_database["action"])

    return action
```

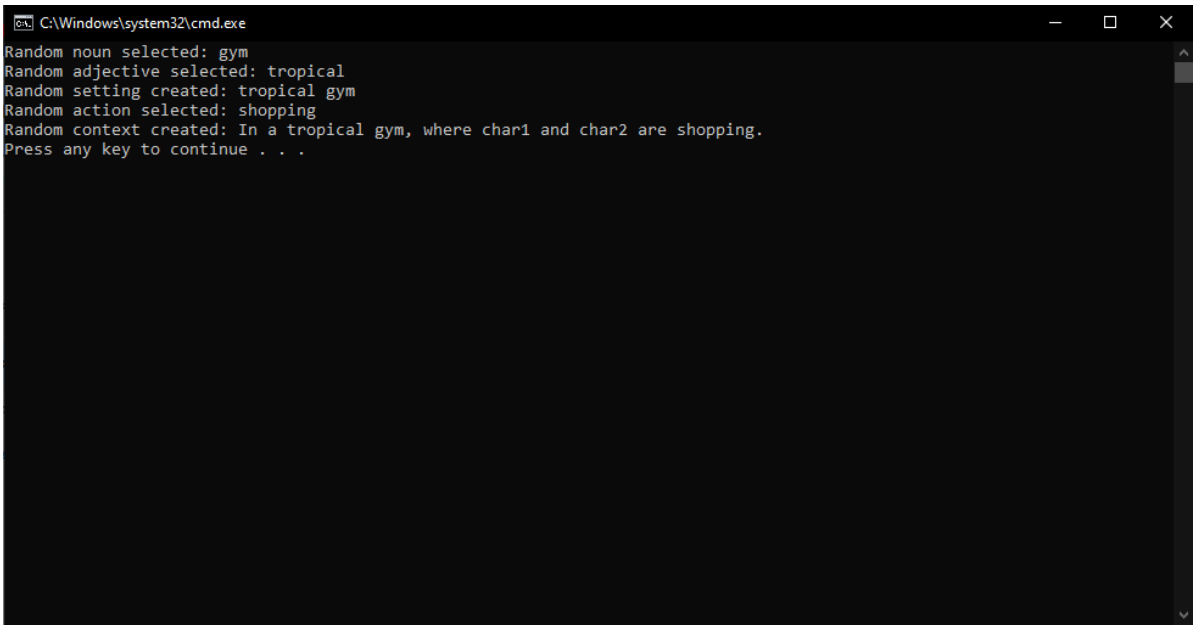
This code right there gets a random action from my database. Now, we put it together like this:

```
def generate_random_context():

    setting = generate_random_setting()
    action = generate_random_action()

    context = f"In a {setting}, where char1 and char2 are {action}"

    return context
```

In this function, I make two variables, one for each randomly generated thing. Then, I make one more variable that gets as value a string that includes text and the variables, giving me the final result in the format I showed before.

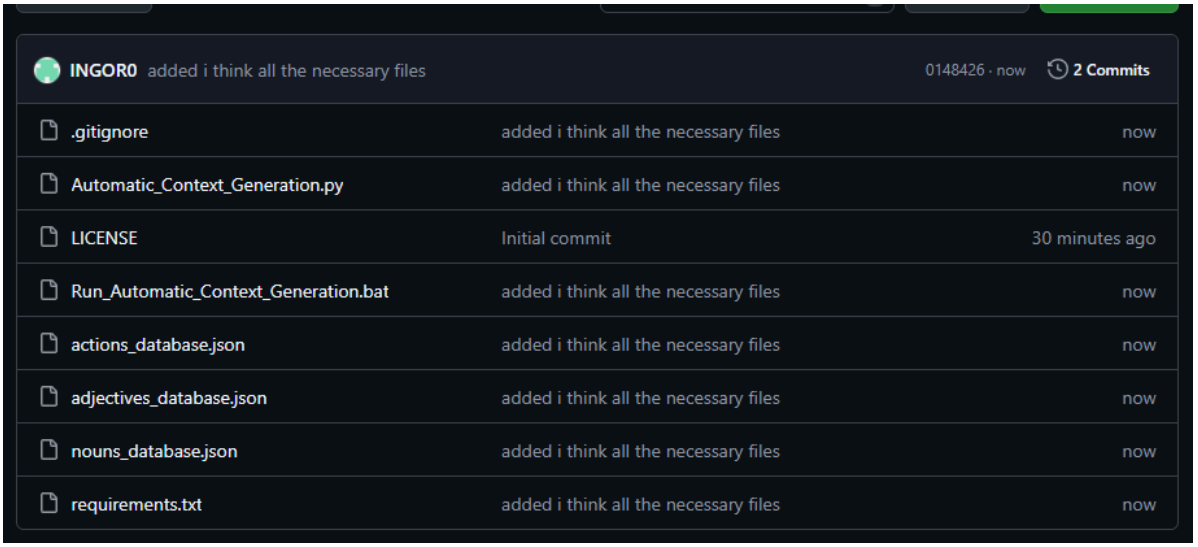
This code, like this, doesn't really do much that we can see. This would be a part of the whole ADMa, which would include a lot of different programs and stuff. But, for the funsies (and to check if it's all good), we can put a print line after each of these blocks to see what the program is doing. For this matter, I will create a duplicate of this code just so I can save the original version without the prints for when I have to use it later.

Although I encountered more issues than I thought, I managed to finish a program able to create a random context given 3 databases (nouns, adjectives and actions). It looks like this once it's finished:



This program is simple and needs those 3 databases, but now I have the possibility to make a random context. Now, I have a lot of options: make the program generate a new context every time it finishes, make the program save it in json format or just text format, etc. This will be used in the bigger and more complex program that I actually want to do, but I will be posting this so you can try it for yourself.

At the end of the day, the start of the next one, I uploaded, or at least I think I uploaded it to **github.**



I have now made **a post in the Python subreddit.** I explain briefly there what my project does and give access to my repository and this document. I'm hoping people will see the disaster I made when doing the repository and will help me to make it look like it should if done correctly. I have stated the name of the automatic context generator to be RaCoGen, but I don't feel like sticking with that.

Anyways, I think my next step is making this program save the contexts generated in a JSON file in a format I can use for my database of contexts. For this, I will rewrite or copy (whatever I find easier) the program into a new one and after texting that everything goes well, I will name it RaCoGen 1.0.

After a few days, I have finished the second version of the context generator program. After thinking about how to describe the process behind that, I decided that it will take less effort to explain the code here rather than in a YouTube video. Let's begin.

I decided to talk about the process after the process itself was done because when working on the first version I was disturbing my workflow by stopping the coding to explain what I was doing here. Not anymore. This time I haven't included a single comment in my code and, in the first place, it was because I was going to first copy and paste it here when explaining it and then I would add the comments before releasing it on github. However, I have already released it, so it will stay that way unless someone specifically asks for a change in that regard.

The first lines of the code are the following:

```
import json
import random
import time
import sys
```

This time, I added the "sys" module so I can make the user close the program by using input when asked for it. You will see what I mean later.

I have updated the databases loading blocks by adding these lines:

```
raise ValueError("Nouns or adjectives database is empty.")
```

```
raise ValueError("Actions database is empty.")
```

With these, I can tell the user if the databases are empty, which is important because the context generation gets the words to fill the blanks from them.

Now we get to my favourite part of the program: saving the contexts in a json file. In fact, it's not only saving them, it is formatting them so that the only thing I have to do when I want more contexts is use the program and forget about writing or formatting on my own. Here's how it's done:

```
contexts_database = 'contexts_database.json'

def save_to_database(context):

    try:

        with open(contexts_database, 'r') as contexts_file:
            existing_data = json.load(contexts_file)

    except (FileNotFoundError, json.JSONDecodeError):

     existing_data = {"contexts": []}

    if context in existing_data["contexts"]:

        print(f"Context already exists: {context}")

    else:

        existing_data["contexts"].append(context)

        with open(contexts_database, 'w') as contexts_file:
            json.dump(existing_data, contexts_file, indent=4)

    print(f"Context: {context}")
```

In this block, I first create a variable just so I don't have to write the json file every time I want to reference it. Then I define a function:

This function tries to open a file named contexts_database.json and, if nothing is found, then it creates one. After that, it checks if the context generated is already saved and, if it is, then it doesn't save it so I don't have duplicates. It also stops if it encounters errors. If the context is already there, it will be printed as "Context already exists: ___"; if not, then it will be printed as "Context: ___".

The next part is more dense and is the part that cost me more work. Let's go into it:

```
def generate_by_time():
def generate_certain_number():
def continue_generating():
def use_the_program():
def program_started():
```

These functions are made in order to make the code cleaner and easier to expand. It is not necessary to do this because I could always just use "if, else" statements to do a lot of things, but by creating a function I only have to do it once.

generate_by_time

This function is referenced when the user chooses to use time as the measuring method for the program to work. When you choose this method, you will be asked to introduce a valid number in seconds, and that number will be the number of seconds that the program will be generating contexts for.

It works with a "try, except" method.