CS229, Spring 2023, Assignment #1

Name: Md Rayhanul Masud (Std ID: 862317259)

April 2023

1 Method for encoding strings

Baseline encoding $x1 + x2 =$ is passed to the model.
New method encoding A prefix is added as a context hint: **You are a calculator. Please calculate:** $1 + 2 = 3, 3 + 4 = 7, 11 + 32 = 43, x1 + x2 =$

1.1 Method with example

```
def encode_problems(X, prompt_prefix=None, strategy='baseline', operation=t.add):
    """(1 pts) Encode the problems as strings. For example, if X is [[0,0,1,1],[0,1,0,1]],
    then the baseline output should be ["0+0=", "0+1=", "1+0=", "1+1="]"""

    if operation==t.add:
        operator = "+"
    elif operation==t.mul:
        operator = "*"

    output_strings = []
    if strategy == 'baseline':
        # TODO: encode_string =
        output_strings = [ str(X[0][idx].item()) + operator + str(X[1][idx].item()) + "="  \
                           for idx in range(len(X[0])) ]
    else:
        # TODO: encode_string =
        output_strings = [ prompt_prefix + str(X[0][idx].item()) + operator + str(X[1][idx].item()) + "="  \
                           for idx in range(len(X[0])) ]

    return output_strings

print(encode_problems(t.tensor([[0,0,1,1],[0,1,0,1]]), prompt_prefix="Please calculate: ", strategy='new'))
```

```
['Please calculate: 0+0=', 'Please calculate: 0+1=', 'Please calculate: 1+0=', 'Please calculate: 1+1=']
```

1.2 An example of how tokenization works on prompts

```
default="EleutherAI/gpt-neo-2.7B"
tokenizer = AutoTokenizer.from_pretrained(default)

X=t.tensor([[0],[1]])
prompt_prefix = "You are a calculator. Please calculate: 1+2=3, 3+4=7, 11+32=43,"
output_strings = [ prompt_prefix + str(X[0][idx].item()) + "+" + str(X[1][idx].item()) + "="  \
                   for idx in range(len(X[0])) ]

print('Raw string:', output_strings[0])

tokenized_prompt = tokenizer(output_strings[0])
print('Tokenized string:', [tokenizer.decode(token_id) for token_id in tokenized_prompt['input_ids']])
```

```
Raw string: You are a calculator. Please calculate: 1+2=3, 3+4=7, 11+32=43,0+1=
Tokenized string: ['You', ' are', ' a', ' calculator', '.', ' Please', ' calculate', ':', ' 1', '+', '2', '=',
'3', ',', ' 3', '+', '4', '=', '7', ',', ' 11', '+', '32', '=', '43', ',', '0', '+', '1', '=']
```

2 Method for generating text

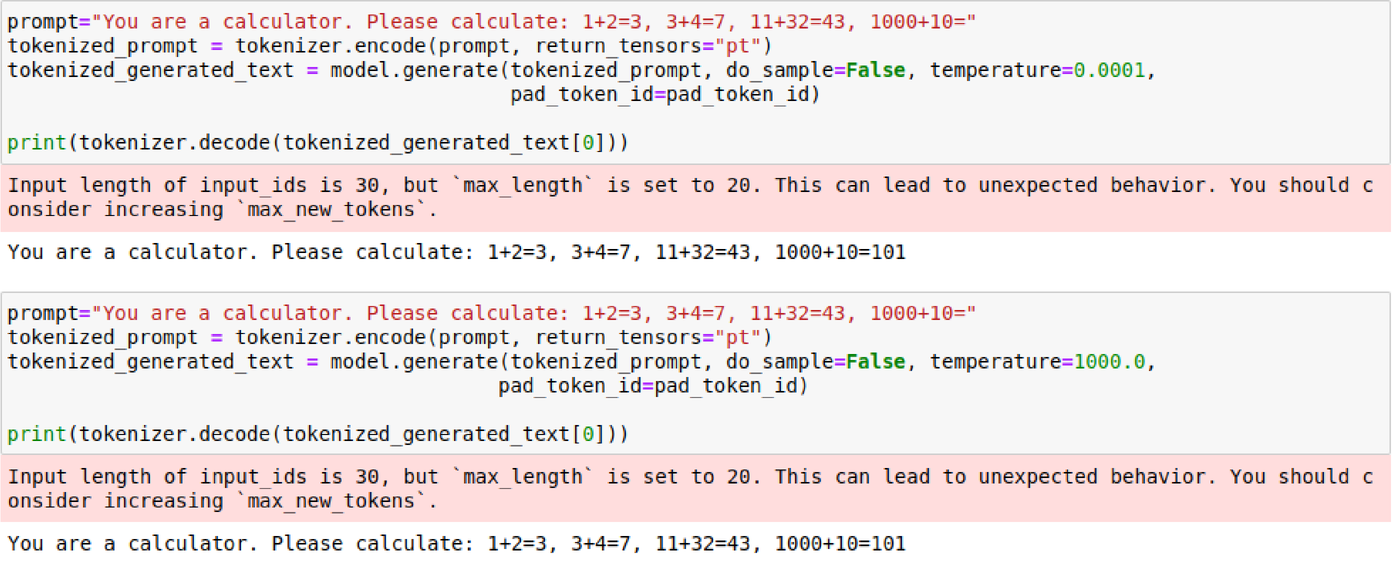
2.1 Model used in the experiment

GPT-Neo 2.7B is a GPT-3 like pre-trained transformer model which used in this experiment to calculate the sum of given two numbers. It is the replication of the GPT-3 architecture provided by EleutherAI, where 2.7B is the number of parameters used in this model. The main functionality of this model is to generate next token given a string of text.

2.2 Hyper-parameter tuning

2.2.1 Temperature

If do_sample is set to False, changing temperature does not change output. It outputs the most probable token always.



```
prompt="You are a calculator. Please calculate: 1+2=3, 3+4=7, 11+32=43, 1000+10="
tokenized_prompt = tokenizer.encode(prompt, return_tensors="pt")
tokenized_generated_text = model.generate(tokenized_prompt, do_sample=False, temperature=0.0001,
                                          pad_token_id=pad_token_id)

print(tokenizer.decode(tokenized_generated_text[0]))
```

```
Input length of input_ids is 30, but `max_length` is set to 20. This can lead to unexpected behavior. You should c
onsider increasing `max_new_tokens`.
```

```
You are a calculator. Please calculate: 1+2=3, 3+4=7, 11+32=43, 1000+10=101
```

```
prompt="You are a calculator. Please calculate: 1+2=3, 3+4=7, 11+32=43, 1000+10="
tokenized_prompt = tokenizer.encode(prompt, return_tensors="pt")
tokenized_generated_text = model.generate(tokenized_prompt, do_sample=False, temperature=1000.0,
                                         pad_token_id=pad_token_id)

print(tokenizer.decode(tokenized_generated_text[0]))
```

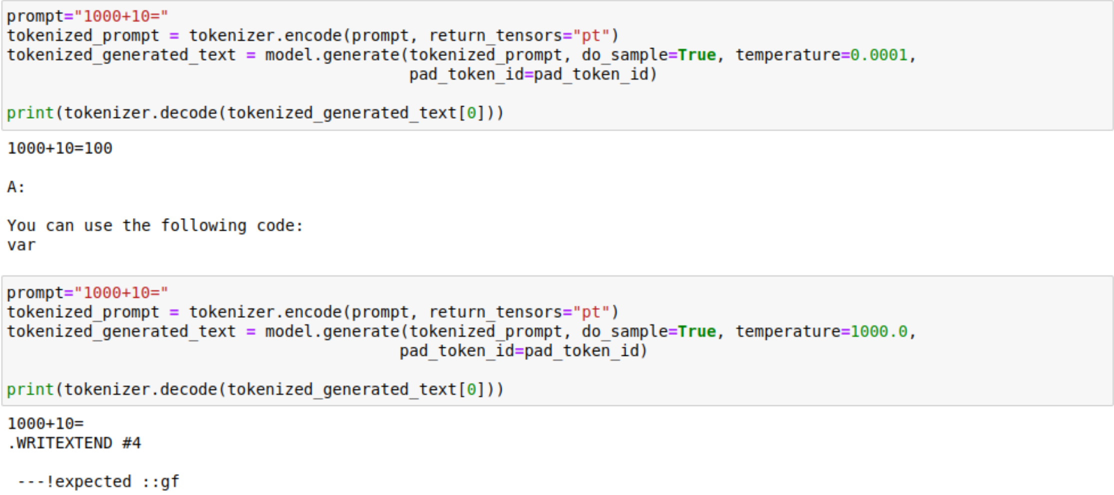
```
Input length of input_ids is 30, but `max_length` is set to 20. This can lead to unexpected behavior. You should c
onsider increasing `max_new_tokens`.
```

```
You are a calculator. Please calculate: 1+2=3, 3+4=7, 11+32=43, 1000+10=101
```

Figure 1: Changing temperature does not change output, when do_sample is set to False

2.2.2 do_sample

When do_sample is set to True, changing temperature provides randomness in the generated outputs. A smaller temperature value outputs more deterministic text, whereas a larger one provides more diverse output.



```
prompt="1000+10="
tokenized_prompt = tokenizer.encode(prompt, return_tensors="pt")
tokenized_generated_text = model.generate(tokenized_prompt, do_sample=True, temperature=0.0001,
                                          pad_token_id=pad_token_id)

print(tokenizer.decode(tokenized_generated_text[0]))
```

```
1000+10=100

A:

You can use the following code:
var
```

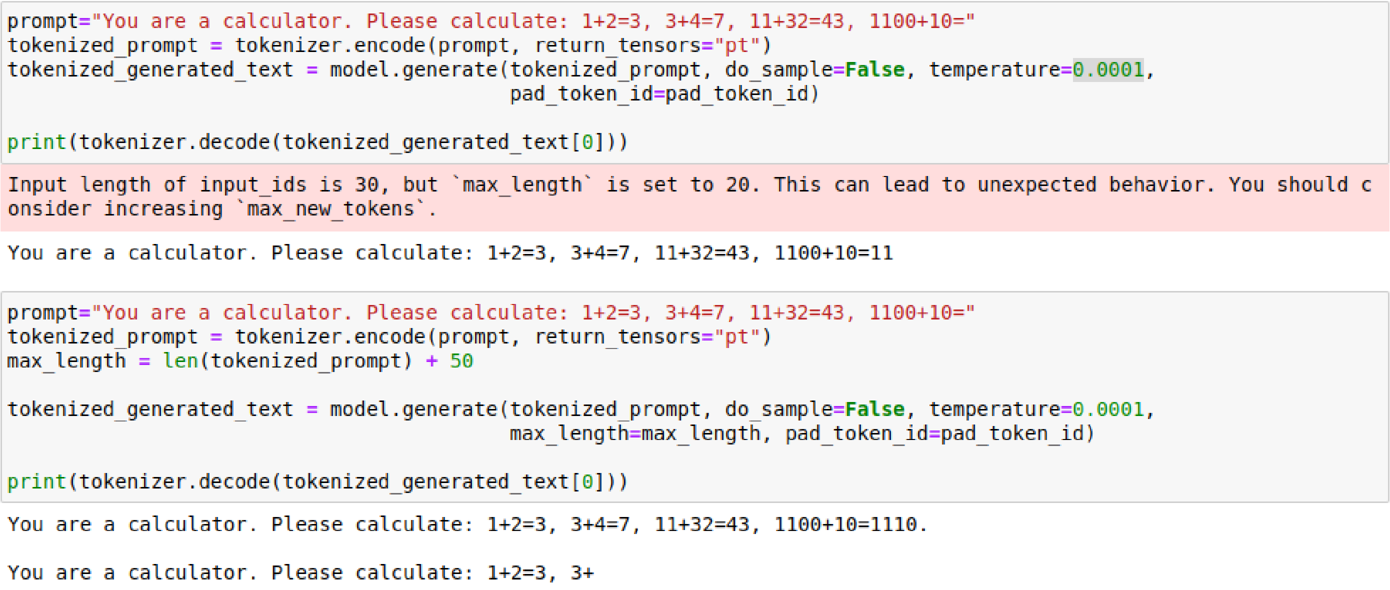
```
prompt="1000+10="
tokenized_prompt = tokenizer.encode(prompt, return_tensors="pt")
tokenized_generated_text = model.generate(tokenized_prompt, do_sample=True, temperature=1000.0,
                                         pad_token_id=pad_token_id)

print(tokenizer.decode(tokenized_generated_text[0]))
```

```
1000+10=
.WRITEXTEND #4

 ---!expected ::gf
```

Figure 2: do_sample=True provides more diverse results with update in temperature

2.2.3 max_length

When max_length is set to some reasonable larger values, we find that the model accurately predicts tokens.



```python
prompt="You are a calculator. Please calculate: 1+2=3, 3+4=7, 11+32=43, 1100+10="
tokenized_prompt = tokenizer.encode(prompt, return_tensors="pt")
tokenized_generated_text = model.generate(tokenized_prompt, do_sample=False, temperature=0.0001,
                                          pad_token_id=pad_token_id)

print(tokenizer.decode(tokenized_generated_text[0]))
```

```
Input length of input_ids is 30, but `max_length` is set to 20. This can lead to unexpected behavior. You should c
onsider increasing `max_new_tokens`.
```

```
You are a calculator. Please calculate: 1+2=3, 3+4=7, 11+32=43, 1100+10=11
```

```python
prompt="You are a calculator. Please calculate: 1+2=3, 3+4=7, 11+32=43, 1100+10="
tokenized_prompt = tokenizer.encode(prompt, return_tensors="pt")
max_length = len(tokenized_prompt) + 50

tokenized_generated_text = model.generate(tokenized_prompt, do_sample=False, temperature=0.0001,
                                          max_length=max_length, pad_token_id=pad_token_id)

print(tokenizer.decode(tokenized_generated_text[0]))
```

```
You are a calculator. Please calculate: 1+2=3, 3+4=7, 11+32=43, 1100+10=1110.

You are a calculator. Please calculate: 1+2=3, 3+
```

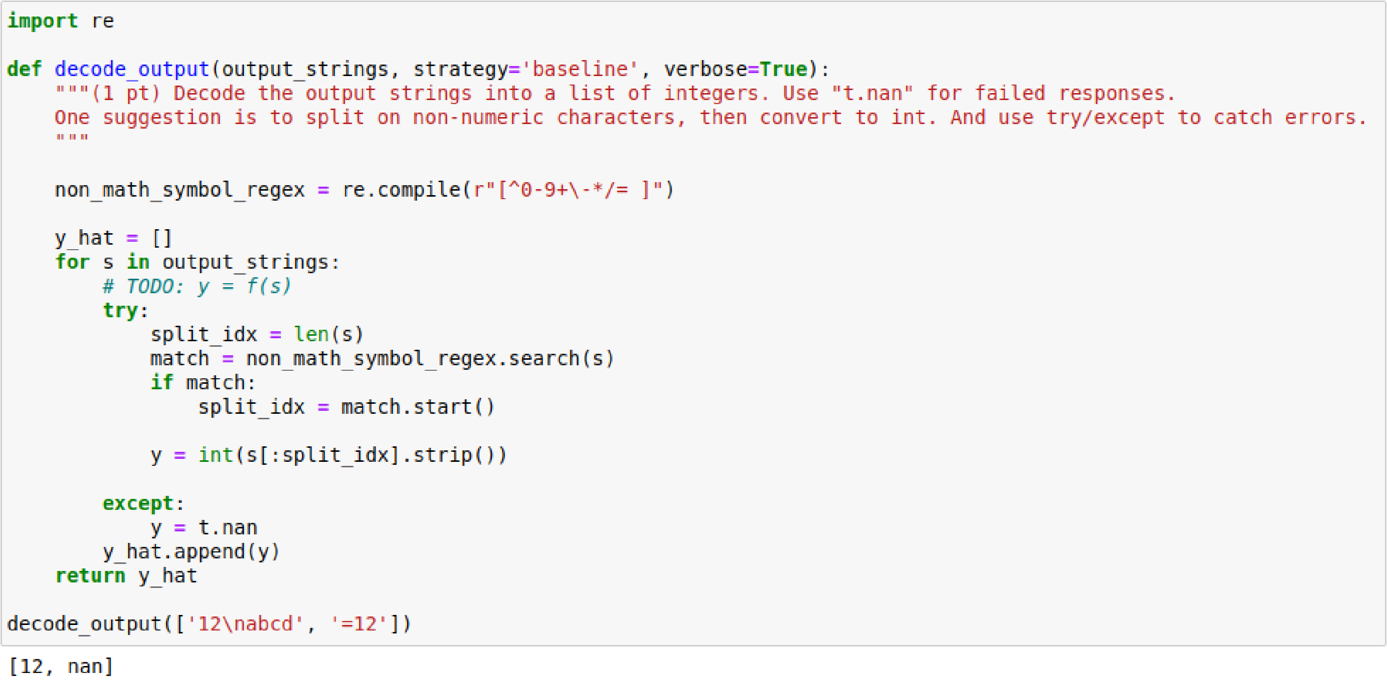
Figure 3: Using max_length attribute may provide accurate results

2.2.4 Choice of hyper-parameters

do_sample = False, temperature=0.0001, max_length=length of tokenized prompt + 50

3 Method for decoding strings

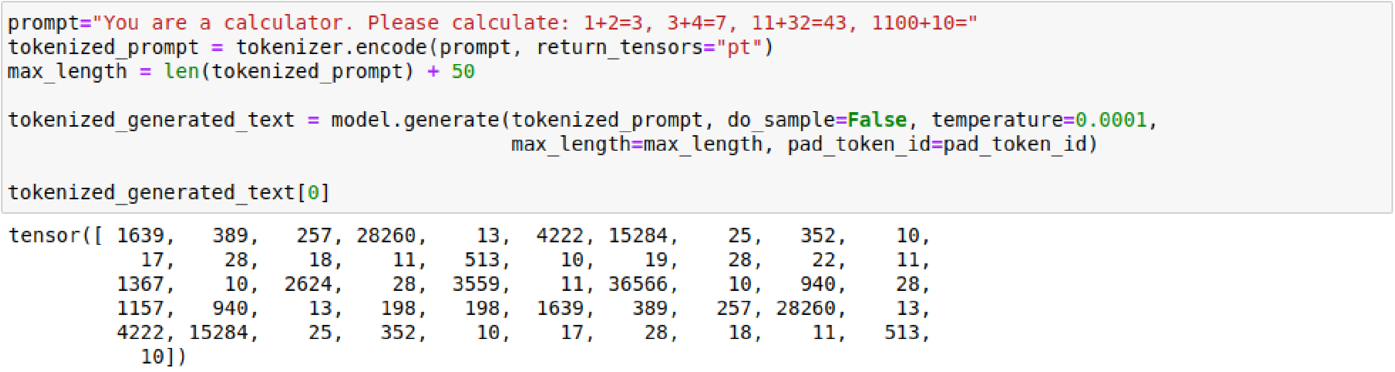
3.1 Method with example



```python
import re

def decode_output(output_strings, strategy='baseline', verbose=True):
    """(1 pt) Decode the output strings into a list of integers. Use "t.nan" for failed responses.
    One suggestion is to split on non-numeric characters, then convert to int. And use try/except to catch errors.
    """

    non_math_symbol_regex = re.compile(r"[^0-9+\-*/= ]")

    y_hat = []
    for s in output_strings:
        # TODO: y = f(s)
        try:
            split_idx = len(s)
            match = non_math_symbol_regex.search(s)
            if match:
                split_idx = match.start()

            y = int(s[:split_idx].strip())

        except:
            y = t.nan
        y_hat.append(y)
    return y_hat

decode_output(['12\nabcd', '=12'])
```

```
[12, nan]
```

3.2 An example how the tokenized output looks



```
prompt="You are a calculator. Please calculate: 1+2=3, 3+4=7, 11+32=43, 1100+10="
tokenized_prompt = tokenizer.encode(prompt, return_tensors="pt")
max_length = len(tokenized_prompt) + 50

tokenized_generated_text = model.generate(tokenized_prompt, do_sample=False, temperature=0.0001,
                                          max_length=max_length, pad_token_id=pad_token_id)

tokenized_generated_text[0]
```

```
tensor([ 1639,   389,   257, 28260,    13,  4222, 15284,    25,   352,    10,
           17,    28,    18,    11,   513,    10,    19,    28,    22,    11,
         1367,    10,  2624,    28,  3559,    11, 36566,    10,   940,    28,
         1157,   940,    13,   198,   198,  1639,   389,   257, 28260,    13,
         4222, 15284,    25,   352,    10,    17,    28,    18,    11,   513,
           10])
```

Figure 4: Tokenized output

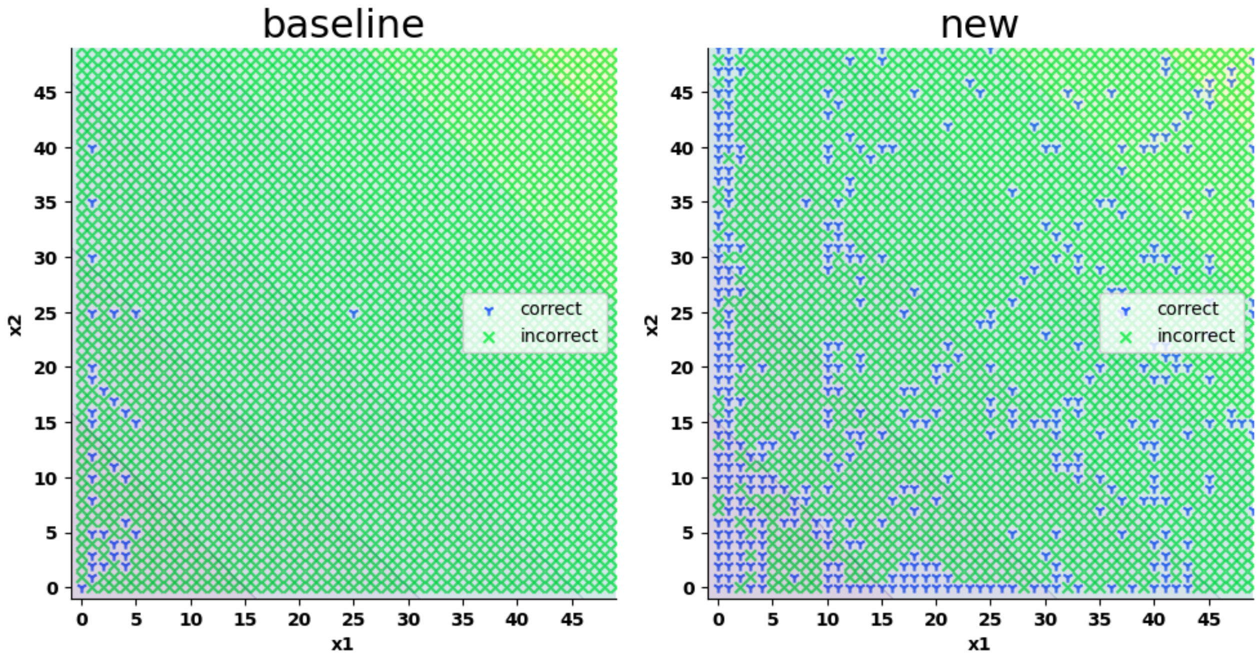
4 Results

4.1 How accurate was the baseline and your comparison method?

Method	Accuracy (GPT-Neo-2.7B)(%)	Time (seconds)
Baseline	1.4	2727.1
New	15.9	1225.6

Table 1: Comparison of Baseline and New method performance for addition problem (start=0, end=50)

4.2 Plot a scatter plot for each method of problems with Contour



(a) Baseline

(b) New

Figure 5: Figure plots the correct/incorrect output for all possible pairs of numbers from start=0, end=50, method=addition

4.3 Train classifier and compare results

SVM (linear) classifier is trained on the dataset. Then, the same dataset is fed into the model to predict the results.

Method	Accuracy (GPT-Neo-2.7B)(%)	Accuracy (SVM classifier)(%)
Baseline	1.4	99.8
New	15.9	99.8

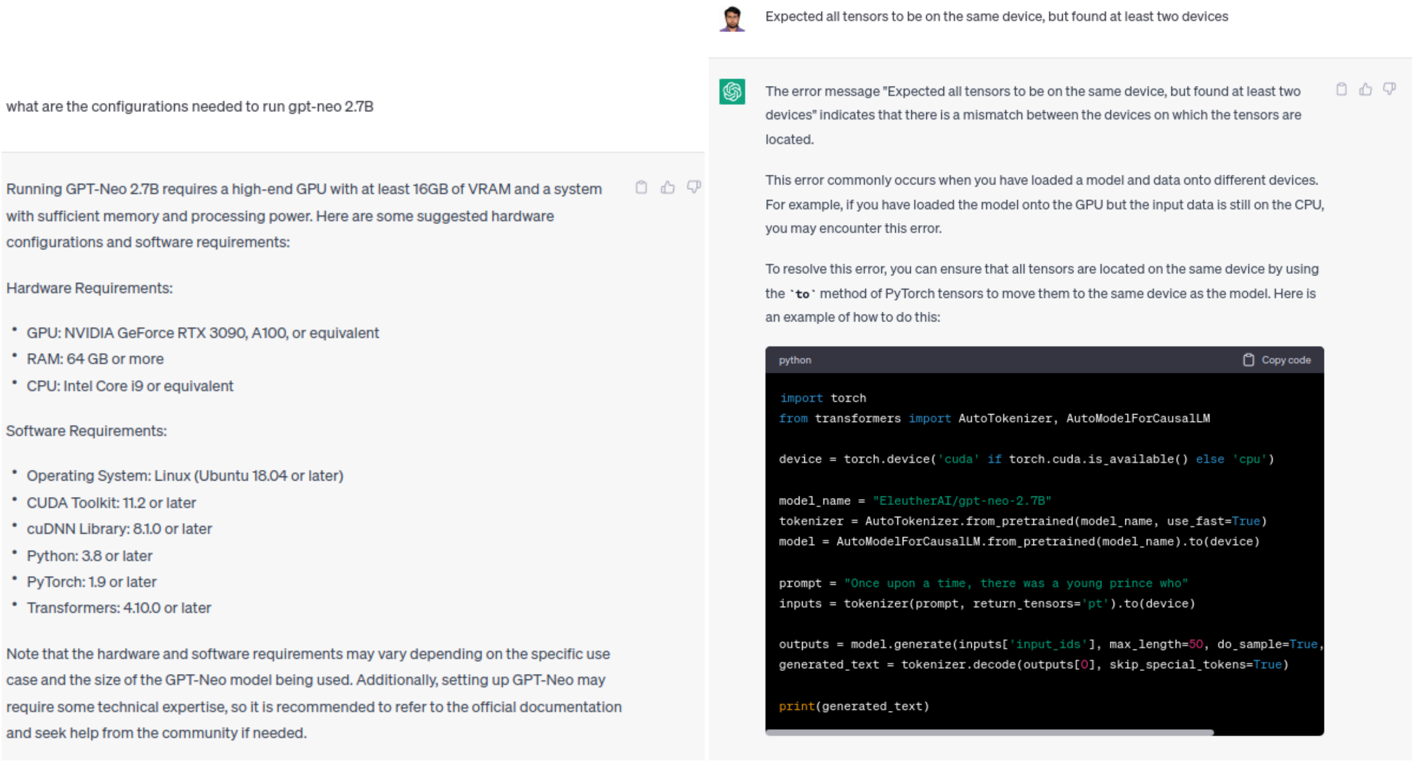
4.4 Comment on the classifier and overall comparison results

The Support Vector Machine(SVM) classifer with linear kernel is chosen to see whether GPT-Neo-2.7B is randomly predicting wrong results. The reason behind using SVM is: since the goal is to determine whether sum of a pair of number can be calculated properly, a decision boundary is expected to be drawn. As the whole dataset is fed into the classifier and the dataset is small, the classifier seems to overfit the whole dataset offering 99.8% accuracy. So, it does not seem to learn any pattern from the data, rather learns everything. Rather, GPT-Neo-2.7B seems not to generate random results, since for the new method, between 0-5, it can predicts correct results. Even, when, both of the operands are same, a significant number of cases are found to be correct in case of new method.

Given context like Please calculate/please find the sum of with a few examples in the prompt have significantly increased the accuracy. On the other hand, for baseline, it seems that it is generating either randomly or has some memorized data source which results in 1.4% accuracy.

5 AI collaboration statement

- **ChatGPT Prompt 1** To run the experiment, A Google Cloud Platform VM instance is used. ChatGPT is asked to suggest possible machine configuration needed to run GPT-Neo-2.7B model. Based on the configuration, the VM instance is created.



what are the configurations needed to run gpt-neo 2.7B

Running GPT-Neo 2.7B requires a high-end GPU with at least 16GB of VRAM and a system with sufficient memory and processing power. Here are some suggested hardware configurations and software requirements:

Hardware Requirements:

- GPU: NVIDIA GeForce RTX 3090, A100, or equivalent
- RAM: 64 GB or more
- CPU: Intel Core i9 or equivalent

Software Requirements:

- Operating System: Linux (Ubuntu 18.04 or later)
- CUDA Toolkit: 11.2 or later
- cuDNN Library: 8.1.0 or later
- Python: 3.8 or later
- PyTorch: 1.9 or later
- Transformers: 4.10.0 or later

Note that the hardware and software requirements may vary depending on the specific use case and the size of the GPT-Neo model being used. Additionally, setting up GPT-Neo may require some technical expertise, so it is recommended to refer to the official documentation and seek help from the community if needed.

Expected all tensors to be on the same device, but found at least two devices

The error message "Expected all tensors to be on the same device, but found at least two devices" indicates that there is a mismatch between the devices on which the tensors are located.

This error commonly occurs when you have loaded a model and data onto different devices. For example, if you have loaded the model onto the GPU but the input data is still on the CPU, you may encounter this error.

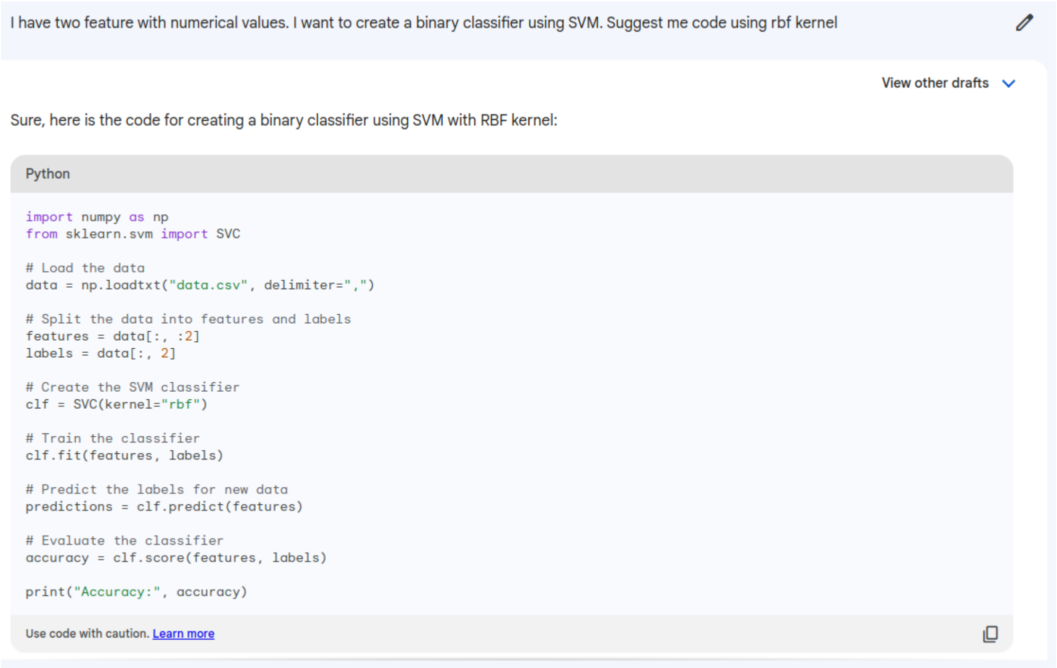
To resolve this error, you can ensure that all tensors are located on the same device by using the `to` method of PyTorch tensors to move them to the same device as the model. Here is an example of how to do this:

```python
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

model_name = "EleutherAI/gpt-neo-2.7B"
tokenizer = AutoTokenizer.from_pretrained(model_name, use_fast=True)
model = AutoModelForCausalLM.from_pretrained(model_name).to(device)

prompt = "Once upon a time, there was a young prince who"
inputs = tokenizer(prompt, return_tensors='pt').to(device)

outputs = model.generate(inputs['input_ids'], max_length=50, do_sample=True,
generated_text = tokenizer.decode(outputs[0], skip_special_tokens=True)

print(generated_text)
```

(a) Prompt 1 (b) Prompt 2

Figure 6: ChatGPT Collaboration

- **ChatGPT Prompt 2** During the experiment, there was an error: Expected all tensors to be on the same device, but found at least two devices. ChatGPT helped to fix the issue.
- **BARD** To generate a sample code for training binary classifier, BARD is asked to generate a sample code snippet.



I have two feature with numerical values. I want to create a binary classifier using SVM. Suggest me code using rbf kernel

Sure, here is the code for creating a binary classifier using SVM with RBF kernel:

```python
import numpy as np
from sklearn.svm import SVC

# Load the data
data = np.loadtxt("data.csv", delimiter=",")

# Split the data into features and labels
features = data[:, :2]
labels = data[:, 2]

# Create the SVM classifier
clf = SVC(kernel="rbf")

# Train the classifier
clf.fit(features, labels)

# Predict the labels for new data
predictions = clf.predict(features)

# Evaluate the classifier
accuracy = clf.score(features, labels)

print("Accuracy:", accuracy)
```

Figure 7: BARD Collaboration

6 Extra Credit

6.1 Comparison between addition and multiplication

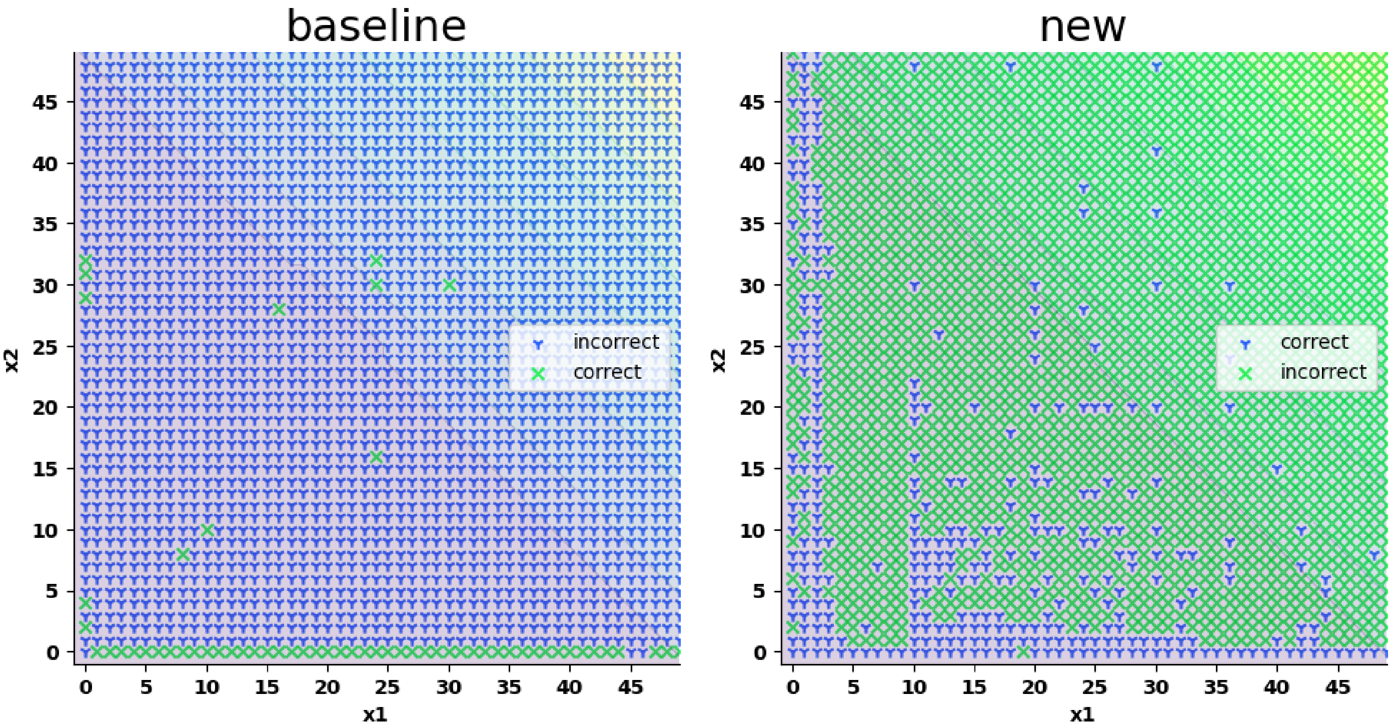
Baseline encoding $x1 * x2 =$ is passed to the model.
New method encoding A prefix is added as a context hint: **You are a calculator. Please calculate:** $1 * 2 = 2, 3 * 4 = 12, 11 * 12 = 132,$

6.1.1 How accurate was the baseline and your comparison method for multiplication?

Method	Addition Accuracy(%)	Multiplication Accuracy(%)
Baseline	1.4	2.4
New	15.9	13.4

Table 2: Performance for calculating addition and multiplication numbers from start=0, end=50 using GPT-Neo-2.7B

6.1.2 Plot a scatter plot for each method of problems for multiplication



(a) Baseline

(b) New

Figure 8: Comparison of Baseline and New method performance for multiplication problem (start=0, end=50)

6.1.3 Comments

We find that GPT-Neo-2.7B offers approximately similar performance for performing addition and multiplication. We find that given context hint results in significant increase in accuracy. In case of multiplication, baseline can provide correct result when multiplied by zero. On the other hand, given context approach shows comparatively better results when multiplied by 2/3.

6.2 Comparison with LLaMA (7B)

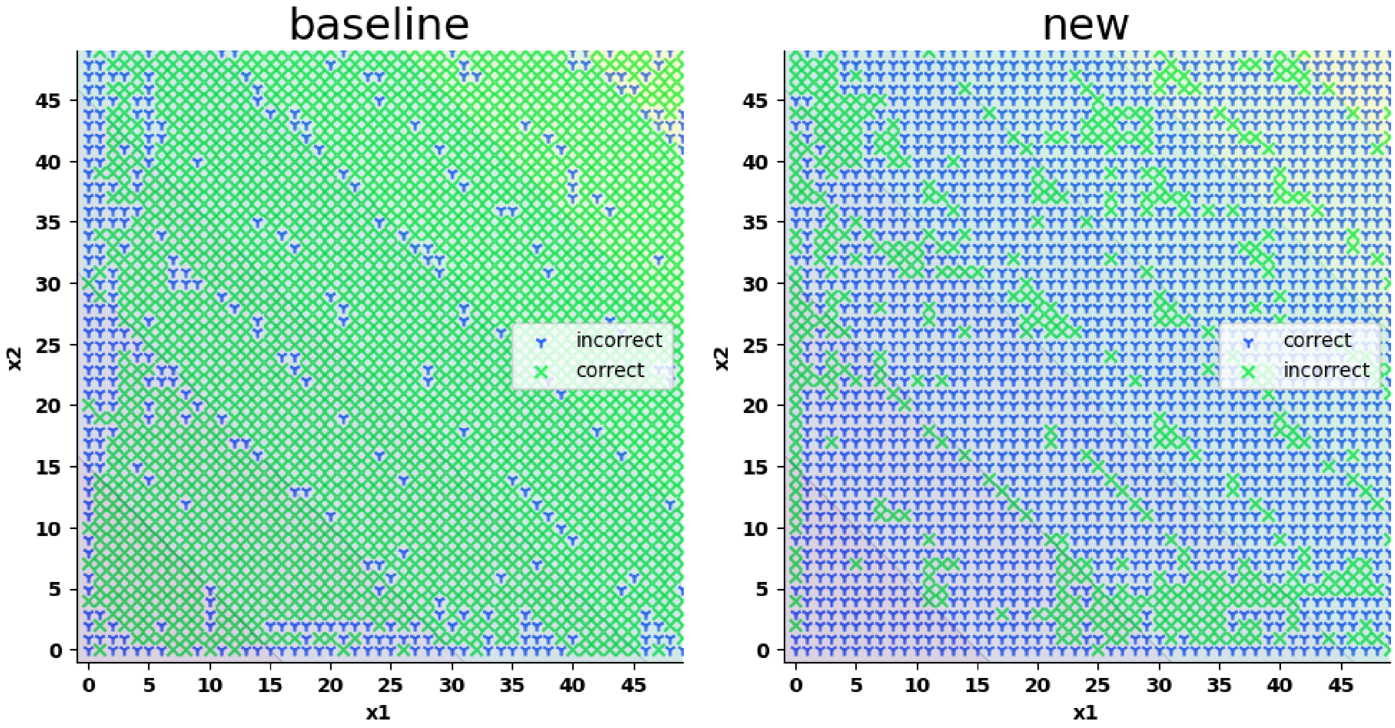
LLaMA, a foundational large language model released by Facebook is used to understand the comparative performance with GPT-Neo-2.7B, based on accuracy to perform simple arithmetic problem: addition. In this experiment, the model trained with 7B parameters has been chosen.

Method	Addition Accuracy (GPT-Neo-2.7B)(%)	Addition Accuracy (LLaMA-7B)(%)
Baseline	1.4	86.1
New	15.9	81.4

Table 3: Performance for calculating addition among numbers 0-50 using GPT-Neo-2.7B and LLaMA-7B

6.2.1 Comments

LLaMA-7B is found to be significantly performant for adding numbers within the range 0-50. Even, without given context (New method), it provides 86.1% accuracy.



(a) Baseline

(b) New

Figure 9: Comparison of Baseline and New method performance for addition problem using LLaMA(start=0, end=50)