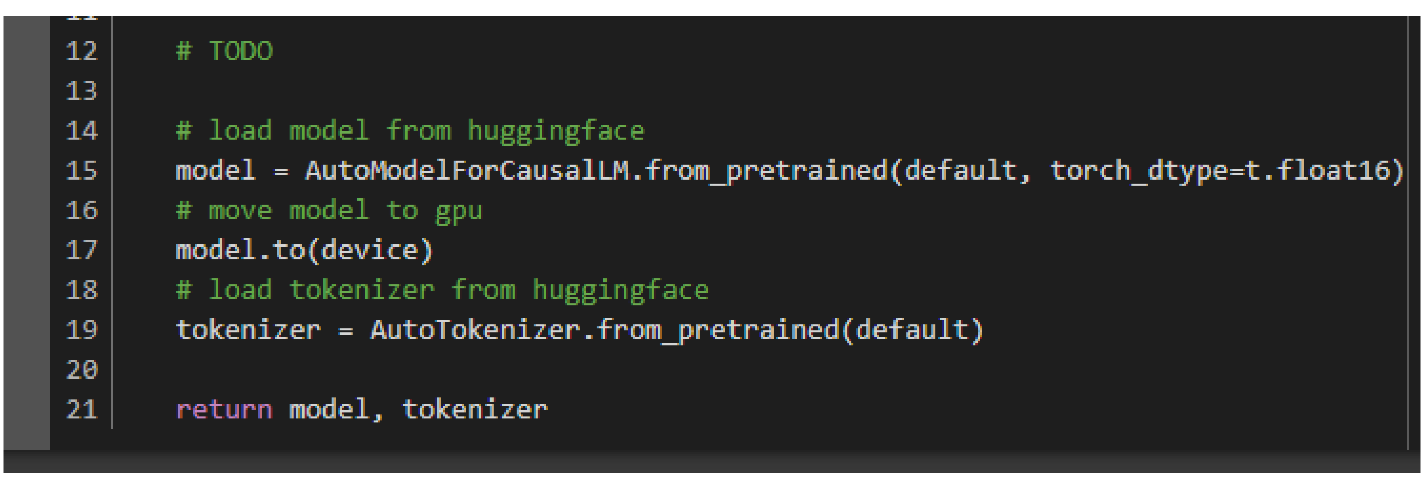
CalcGPT Writeup

Name : Yash Aggarwal
SID: 862333037
NetId: yagga004

Problem Statement:
To run inference on Large Language models and check whether they can perform basic mathematical operations. The operation for this test was '**Addition'** and the LLMs used were '**EleutherAI/gpt-neo-1.3B'** and **'facebook/opt-2.7b'.** After documenting the results, I ran a classifier to check whether the LLMs just randomly predict a token or do they understand mathematical operations.

Tasks.
1. Download and load a LLM using hugging face
2. Download a load a tokenizer for the model
3. Create a dataset X,y where X is input such that shape of X is 2,n where n is number of samples and xi is ith sample and xi[0] is x1 and xi[1] is x2 and y = ans such that yi = xi[0] + xi[1]
4. Encode the dataset into english problem statements in different strategies to make them better understandable by the model
5. Run inference on the model. May need to run them in batches as colab may not support large batch sizes
6. From the outputs, remove the input to get the LLMs answer.
7. Use Regex to get the first number from the answer and store it as yPred
8. Repeat this process for multiple encoding strategies
9. Analyze the results

Downloading the Model and Tokenizer



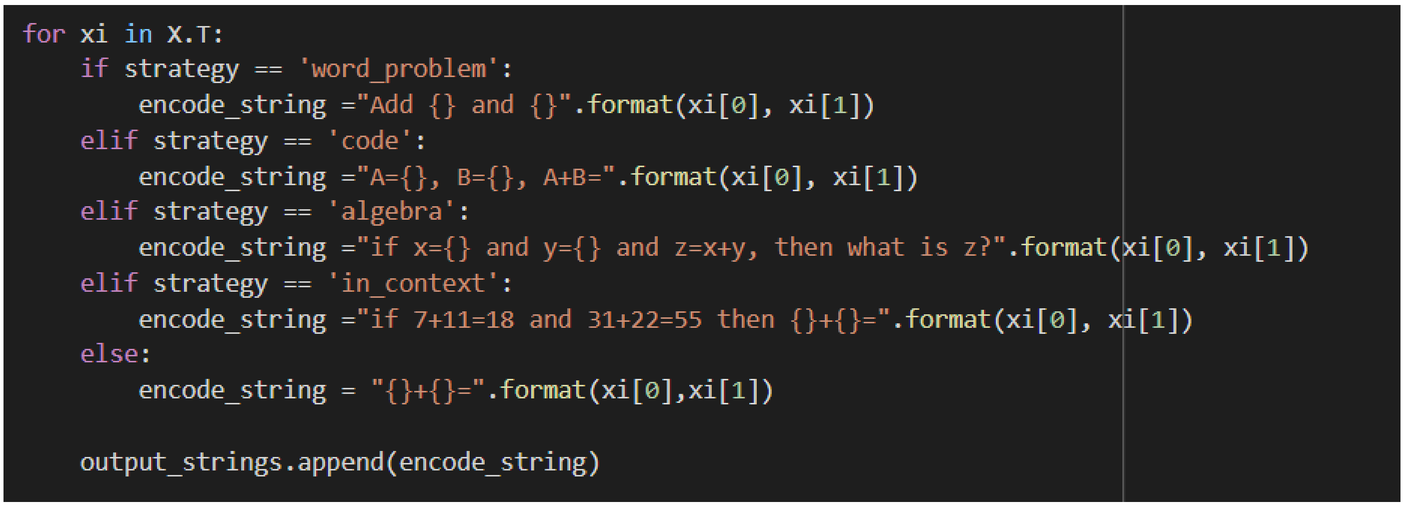
```
    # TODO

    # load model from huggingface
    model = AutoModelForCausalLM.from_pretrained(default, torch_dtype=t.float16)
    # move model to gpu
    model.to(device)
    # load tokenizer from huggingface
    tokenizer = AutoTokenizer.from_pretrained(default)

    return model, tokenizer
```

Encode the Strings

Here I ttook a datasample from the input matrix and generated an input string. For Example and datasample 5,6 is encoded as ‘5+6=’

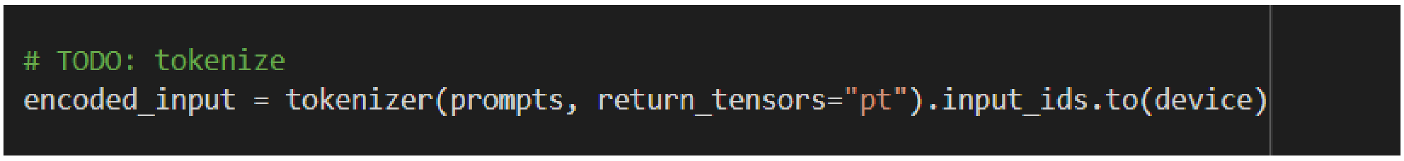
Code

Iterate over the dataset and generate string in required strategy



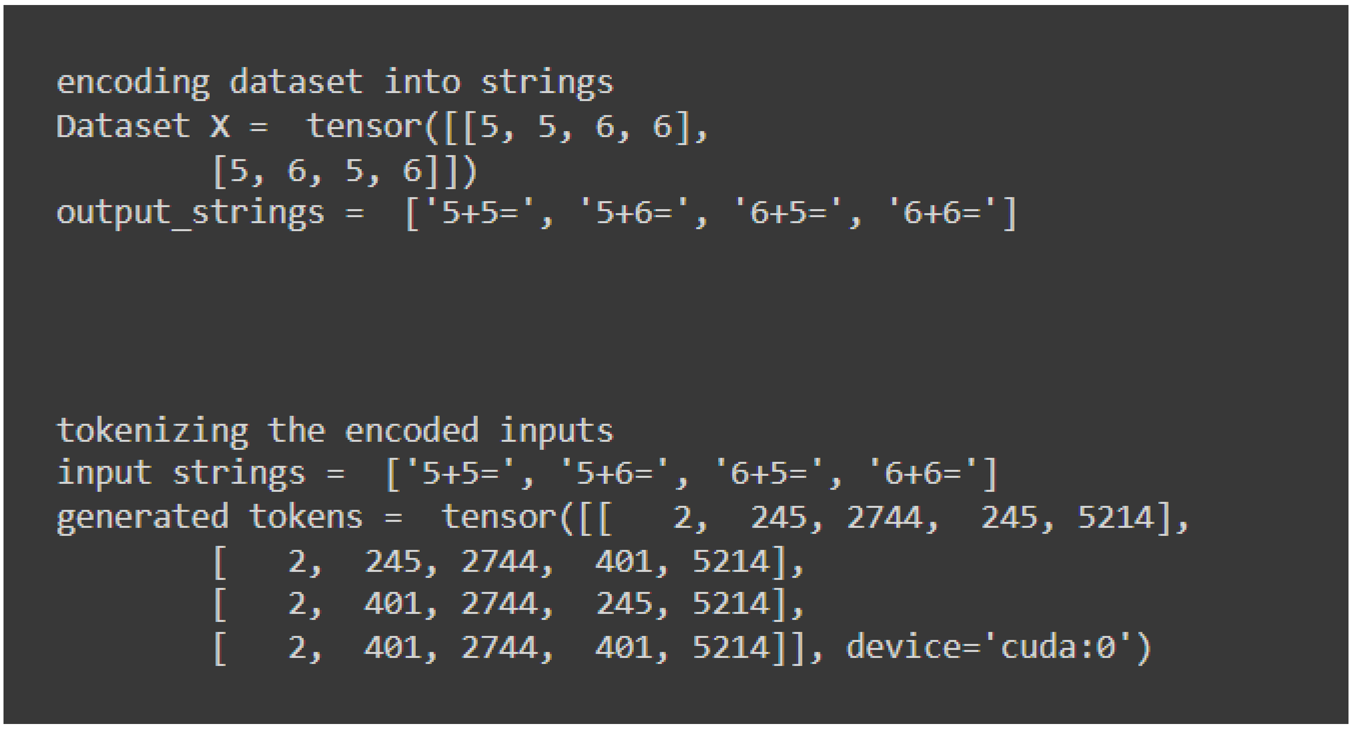
```
for xi in X.T:
    if strategy == 'word_problem':
        encode_string ="Add {} and {}".format(xi[0], xi[1])
    elif strategy == 'code':
        encode_string ="A={}, B={}, A+B=".format(xi[0], xi[1])
    elif strategy == 'algebra':
        encode_string ="if x={} and y={} and z=x+y, then what is z?".format(xi[0], xi[1])
    elif strategy == 'in_context':
        encode_string ="if 7+11=18 and 31+22=55 then {}+{}=".format(xi[0], xi[1])
    else:
        encode_string = "{}+{}=".format(xi[0],xi[1])

    output_strings.append(encode_string)
```

Pass these strings to a tokenizer



```
# TODO: tokenize
encoded_input = tokenizer(prompts, return_tensors="pt").input_ids.to(device)
```

These sentences are converted to tokens to be passed into the LLM
For Example, the sentence ‘5+6=’ is tokenized as 2, 245, 2744, 401, 5214.
Note, the token at the start ‘2’ is for the start token ‘</s>’.



```
encoding dataset into strings
Dataset X =  tensor([[5, 5, 6, 6],
        [5, 6, 5, 6]])
output_strings =  ['5+5=', '5+6=', '6+5=', '6+6=']




tokenizing the encoded inputs
input strings =  ['5+5=', '5+6=', '6+5=', '6+6=']
generated tokens =  tensor([[   2,  245, 2744,  245, 5214],
        [   2,  245, 2744,  401, 5214],
        [   2,  401, 2744,  245, 5214],
        [   2,  401, 2744,  401, 5214]], device='cuda:0')
```

Generating the Text

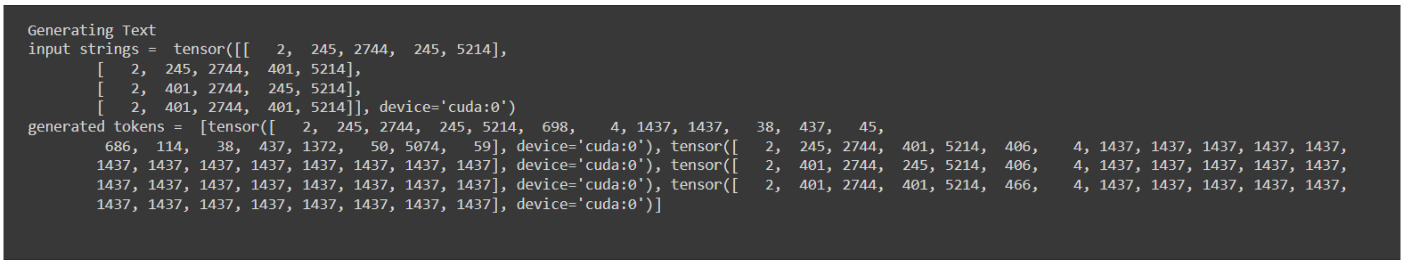
To generate the text we use ‘**EleutherAI/gpt-neo-1.3B’** and **‘facebook/opt-2.7b’.**

EleutherAI/gpt-neo-1.3B is replication of the GPT-3 architecture. GPT-Neo refers to the class of models, while 1.3B represents the number of parameters of this particular pre-trained model.
GPT-Neo was trained as an autoregressive language model. This means that its core functionality is taking a string of text and predicting the next token. While language models are widely used for tasks other than this, there are a lot of unknowns with this work.

facebook/opt-2.7b or Facebook’s OPT was predominantly pretrained with English text, but a small amount of non-English data is still present within the training corpus via CommonCrawl. The model was pre trained using a causal language modeling (CLM) objective. OPT belongs to the same family of decoder-only models like GPT-3. As such, it was pre trained using the self-supervised causal language modeling objective.
For evaluation, OPT follows GPT-3 by using their prompts and overall experimental setup.

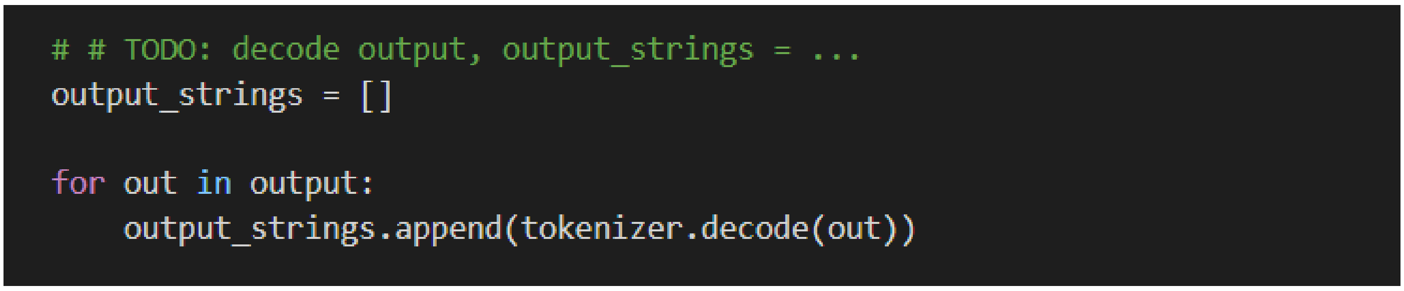
The default hyperparameters are used without sampling or changing the temperature settings as changing them resulted in absolutely wrong results.

The tokens are passed to the model and the model generates next tokens to complete the sentence. These tokens can be seen below.
Here input strings is the tokenized input to model and output is the models generated response, in tokenized form



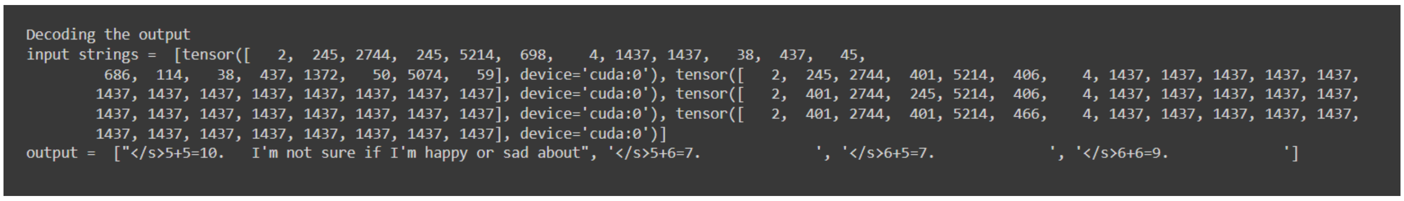
```
Generating Text
input strings =  tensor([[   2,  245, 2744,  245, 5214],
        [   2,  245, 2744,  401, 5214],
        [   2,  401, 2744,  245, 5214],
        [   2,  401, 2744,  401, 5214]], device='cuda:0')
generated tokens =  [tensor([   2,  245, 2744,  245, 5214,  698,    4, 1437, 1437,   38,  437,   45,
         686,  114,   38,  437, 1372,   50, 5074,   59], device='cuda:0'), tensor([   2,  245, 2744,  401, 5214,  406,    4, 1437, 1437, 1437, 1437, 1437,
        1437, 1437, 1437, 1437, 1437, 1437, 1437, 1437], device='cuda:0'), tensor([   2,  401, 2744,  245, 5214,  406,    4, 1437, 1437, 1437, 1437, 1437,
        1437, 1437, 1437, 1437, 1437, 1437, 1437, 1437], device='cuda:0'), tensor([   2,  401, 2744,  401, 5214,  466,    4, 1437, 1437, 1437, 1437, 1437,
        1437, 1437, 1437, 1437, 1437, 1437, 1437, 1437], device='cuda:0')]
```

Decoding String



```python
# # TODO: decode output, output_strings = ...
output_strings = []

for out in output:
    output_strings.append(tokenizer.decode(out))
```

The model output is passed through the tokenizer again and decoded.
Here input string is the output of model and input to tokenizer.decode function
The output is the decoder’s output

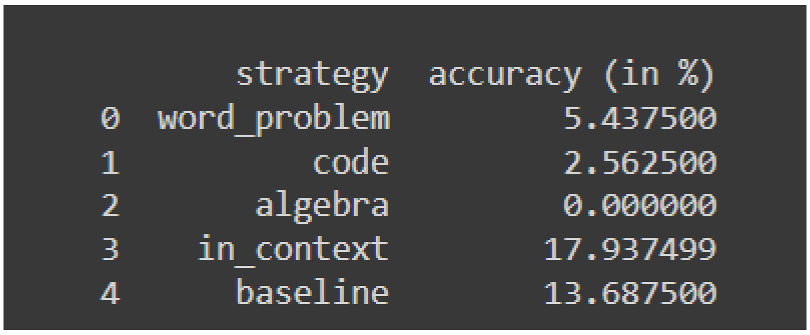


```
Decoding the output
input strings =  [tensor([   2,  245, 2744,  245, 5214,  698,    4, 1437, 1437,   38,  437,   45,
          686,  114,   38,  437, 1372,   50, 5074,   59], device='cuda:0'), tensor([   2,  245, 2744,  401, 5214,  406,    4, 1437, 1437, 1437, 1437, 1437,
         1437, 1437, 1437, 1437, 1437, 1437, 1437, 1437], device='cuda:0'), tensor([   2,  401, 2744,  245, 5214,  406,    4, 1437, 1437, 1437, 1437, 1437,
         1437, 1437, 1437, 1437, 1437, 1437, 1437, 1437], device='cuda:0'), tensor([   2,  401, 2744,  401, 5214,  466,    4, 1437, 1437, 1437, 1437, 1437,
         1437, 1437, 1437, 1437, 1437, 1437, 1437, 1437], device='cuda:0')]
output =  ["</s>5+5=10.   I'm not sure if I'm happy or sad about", '</s>5+6=7.              ', '</s>6+5=7.              ', '</s>6+6=9.              ']
```

Results

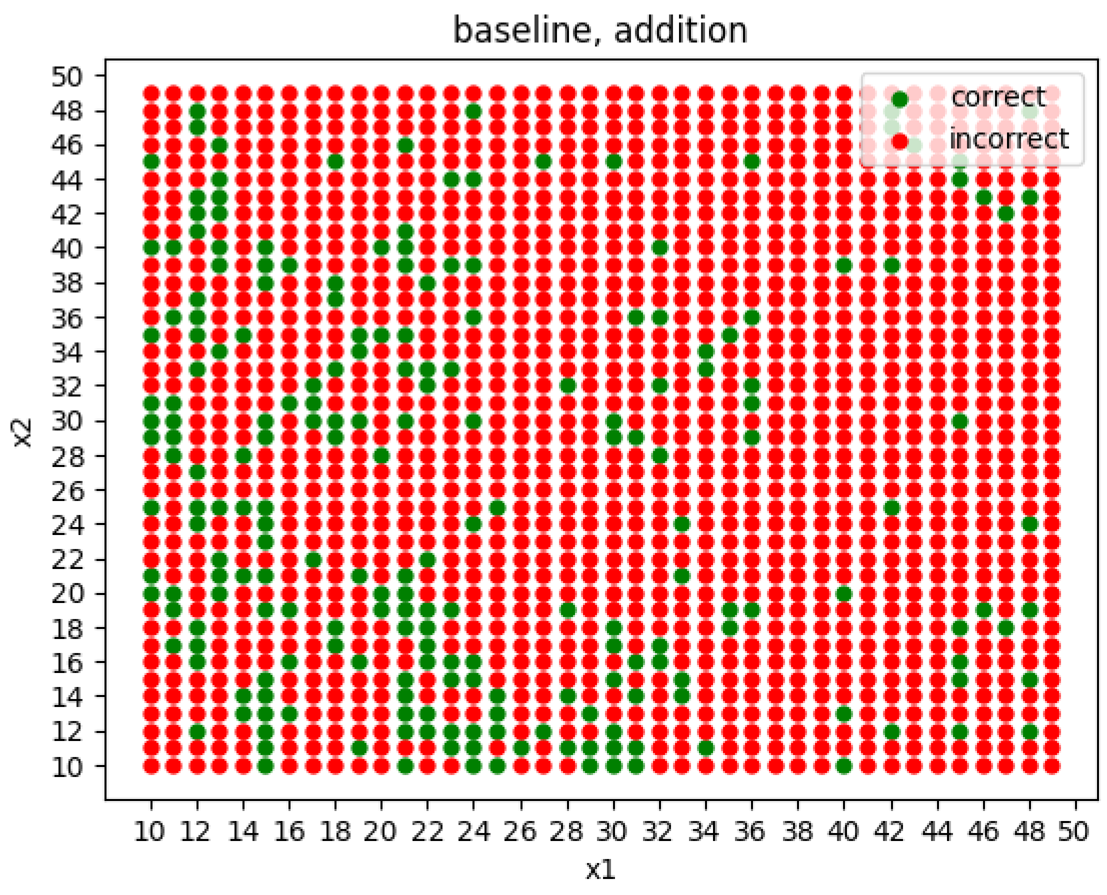
Accuracy

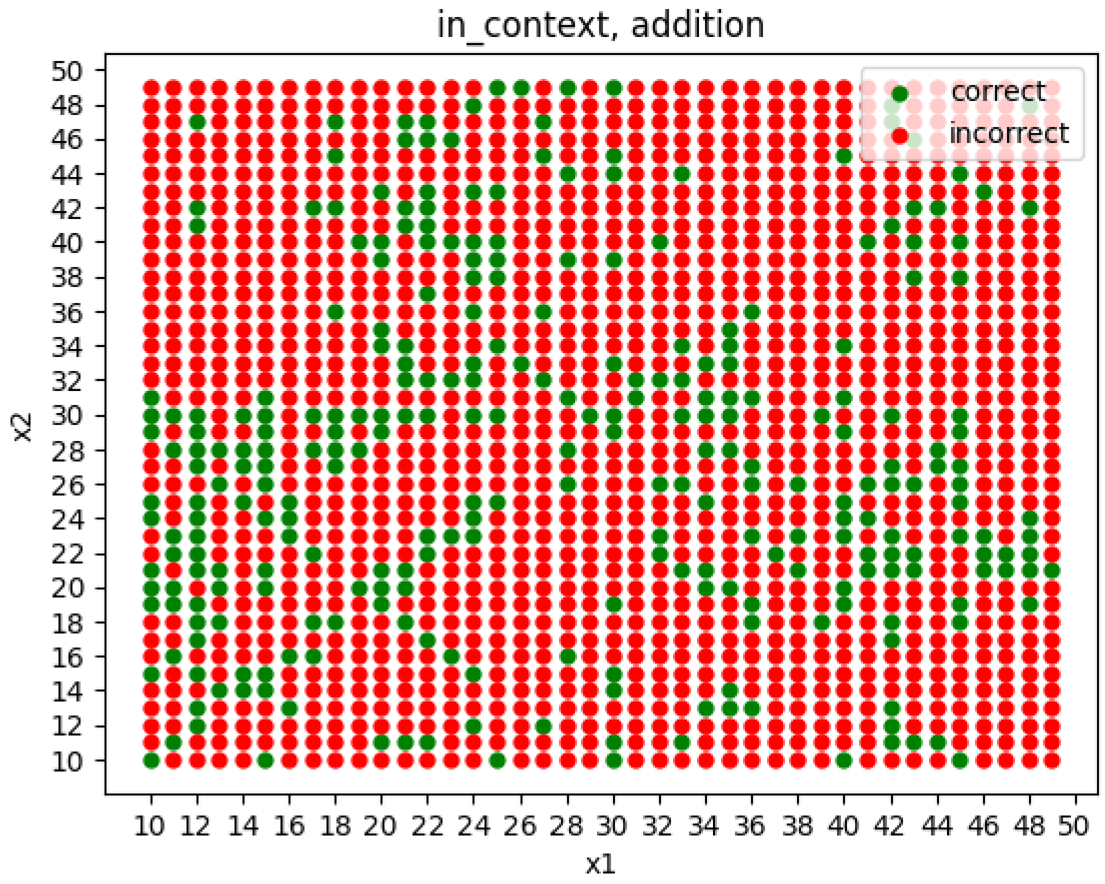
Comparing the various strategies I have the following results.



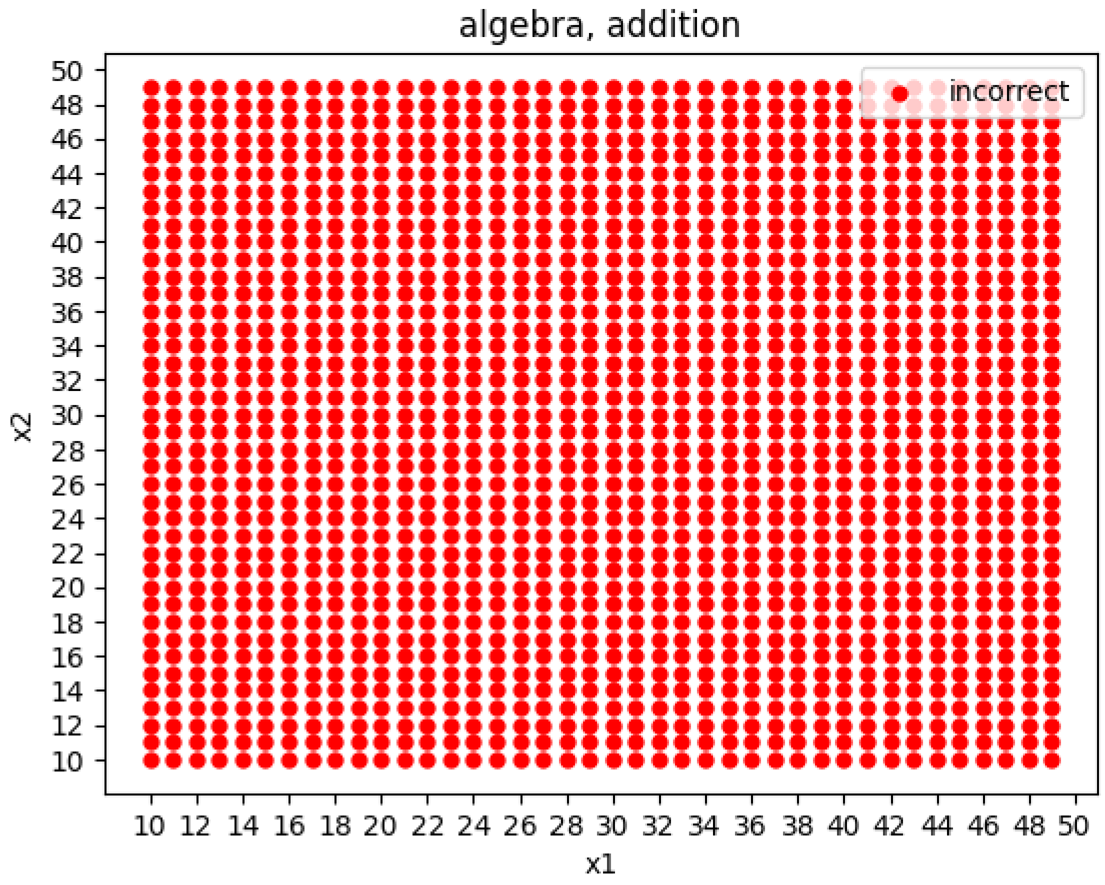
```
      strategy  accuracy (in %)
0  word_problem         5.437500
1          code         2.562500
2       algebra         0.000000
3    in_context        17.937499
4      baseline        13.687500
```

Scatter Plots

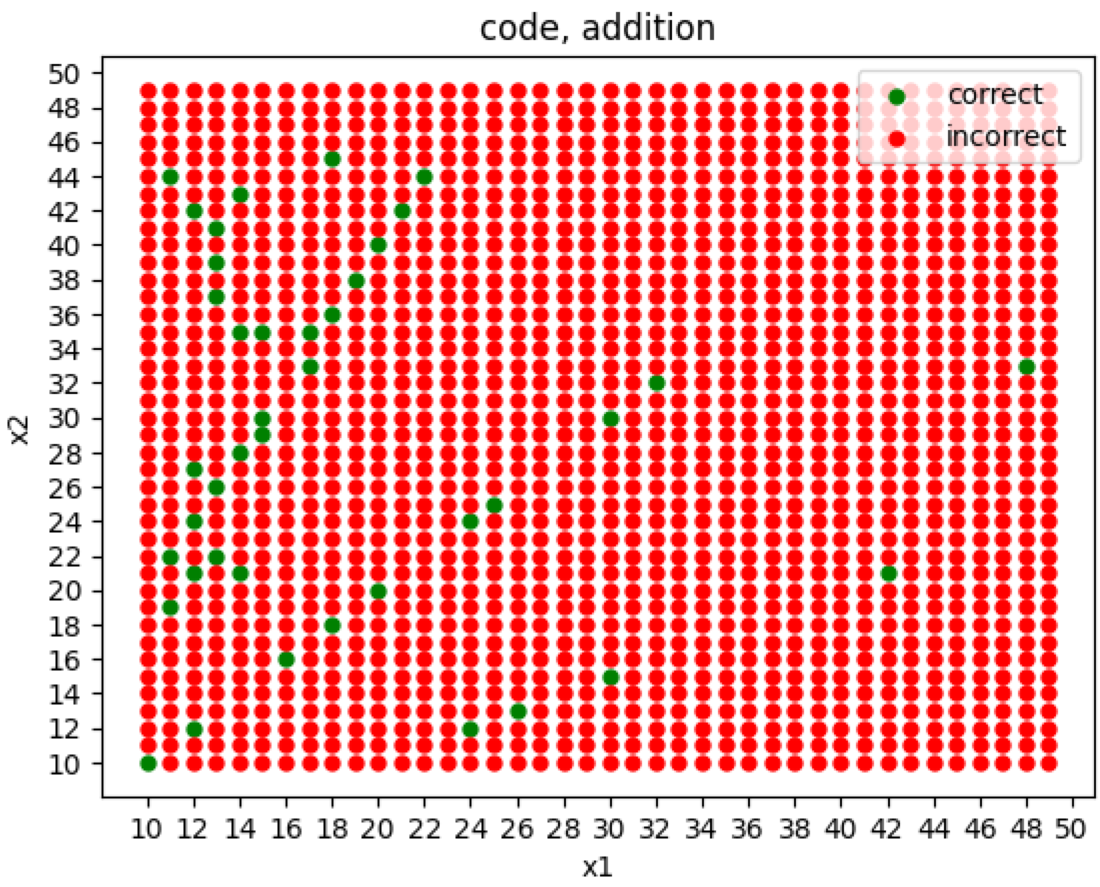


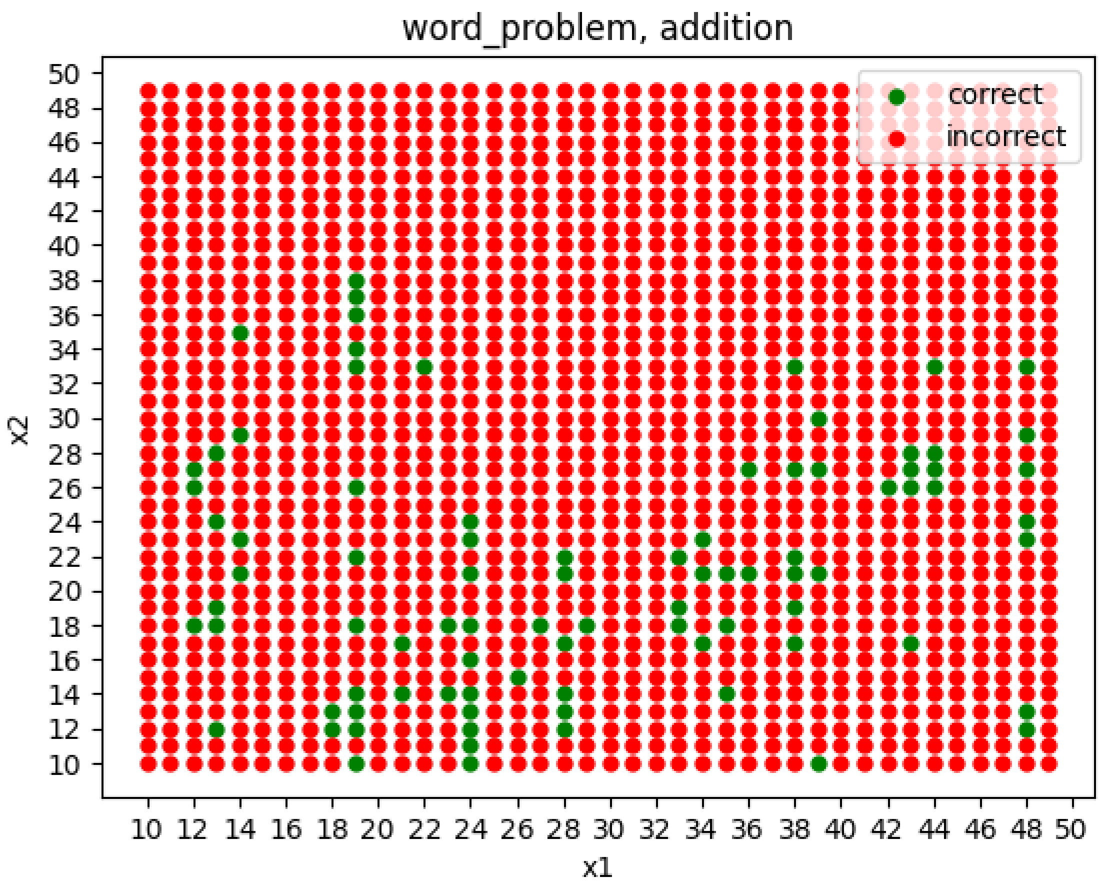


in_context, addition
correct
incorrect
x1
x2



algebra, addition
incorrect
x1
x2



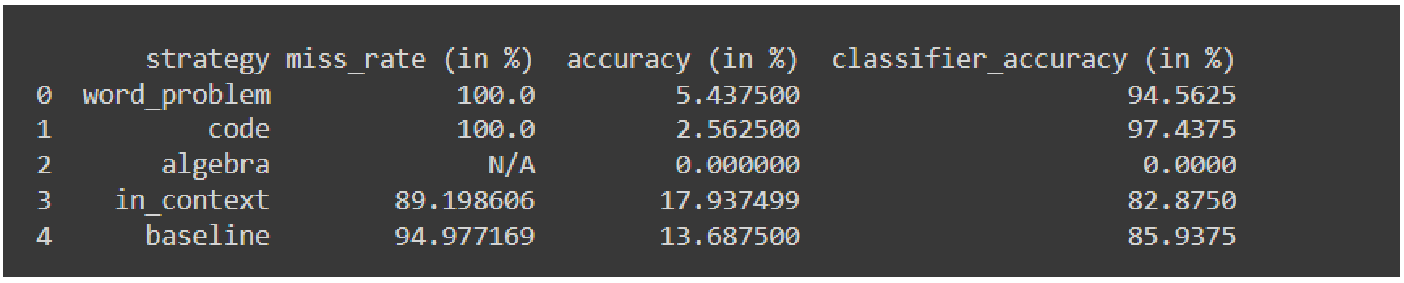


Classifier
Looking at the spread of the plots, it seems that the LLM predicts most of the answers incorrectly. Thus using a linear classifier would be of no use. So I trained a Multi Layer Perceptron using sklearn with hidden layers as 64,64,64,64,32,16,8 and 'relu' activation function for 1000 iterations.

Classifier Accuracy

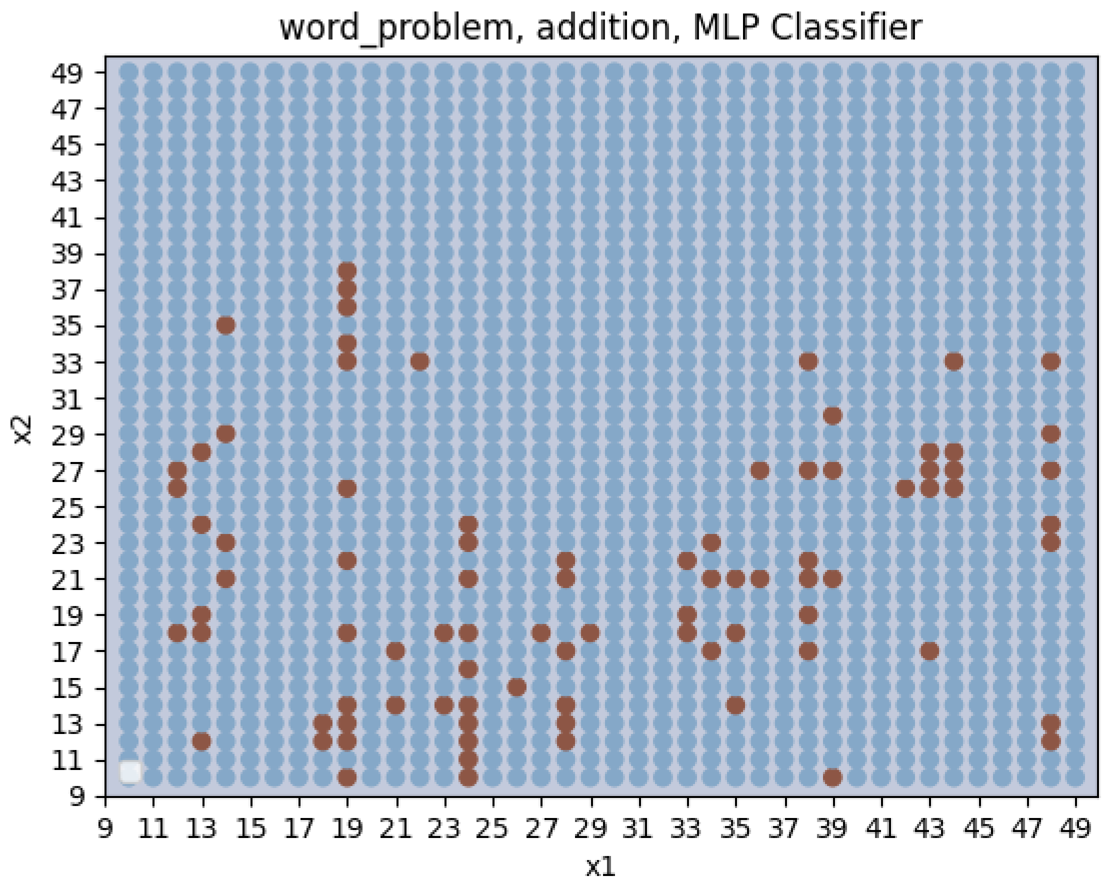
As we have imbalance classes and the classifier would just give out all answers as incorrect and still have high accuracy, I have used miss rate as the metric of evaluation. High miss rate would mean that the classifier incorrectly labels both correct and incorrect classes.

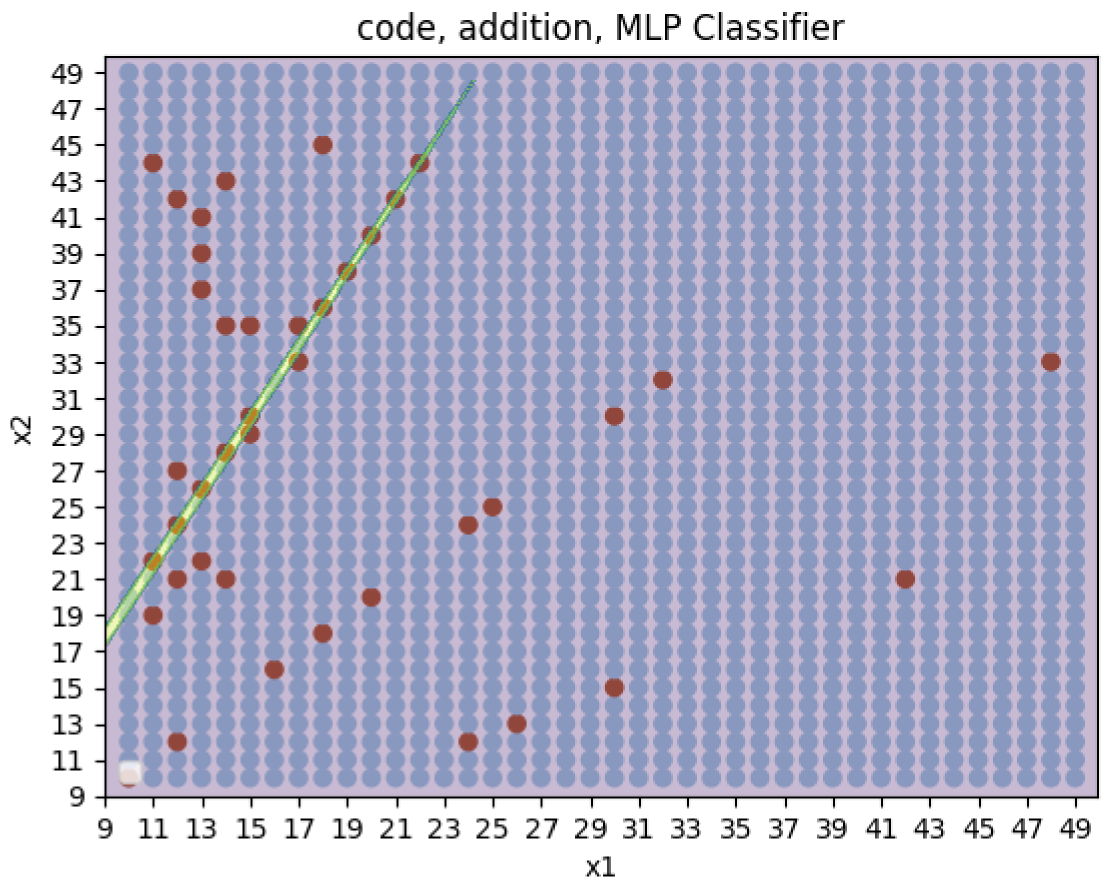
As we can see from the results, even though we have high accuracy, we have an even higher miss rate. This means that the classifier we trained is completely random and this means that the LLM is completely random when predicting the answers.

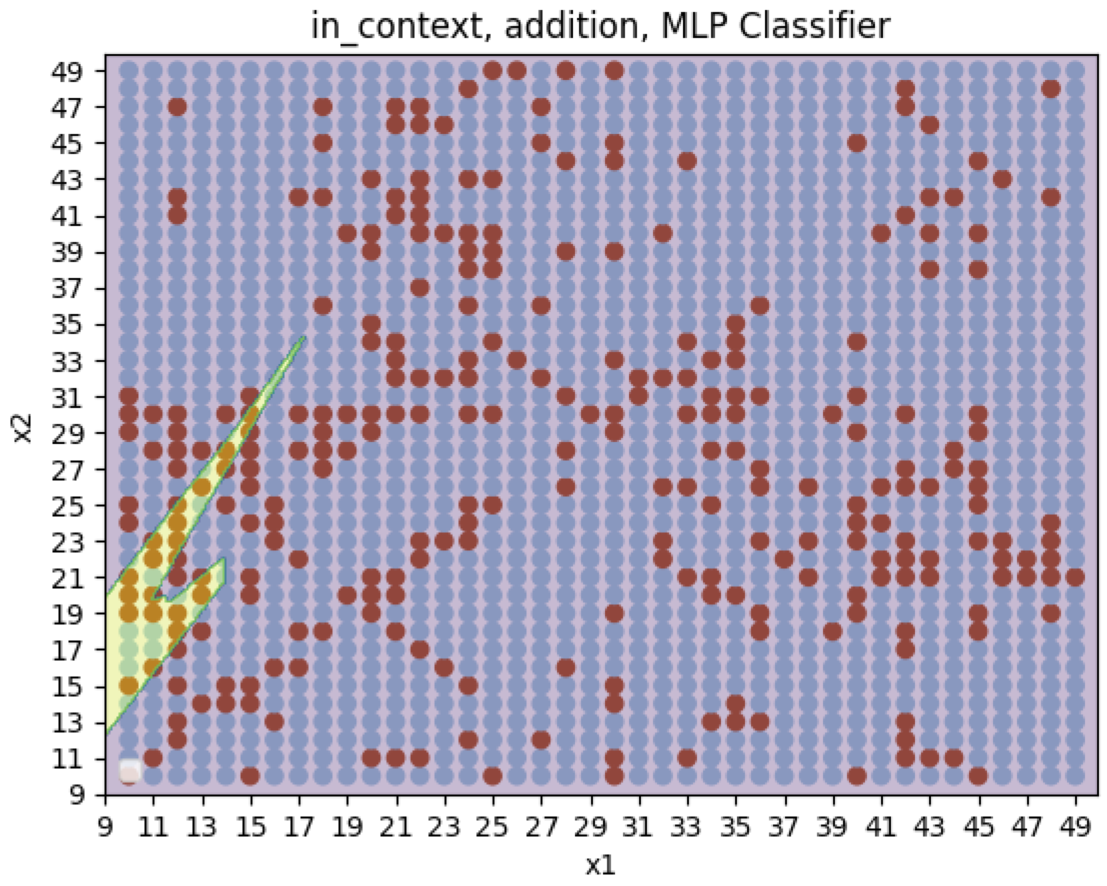


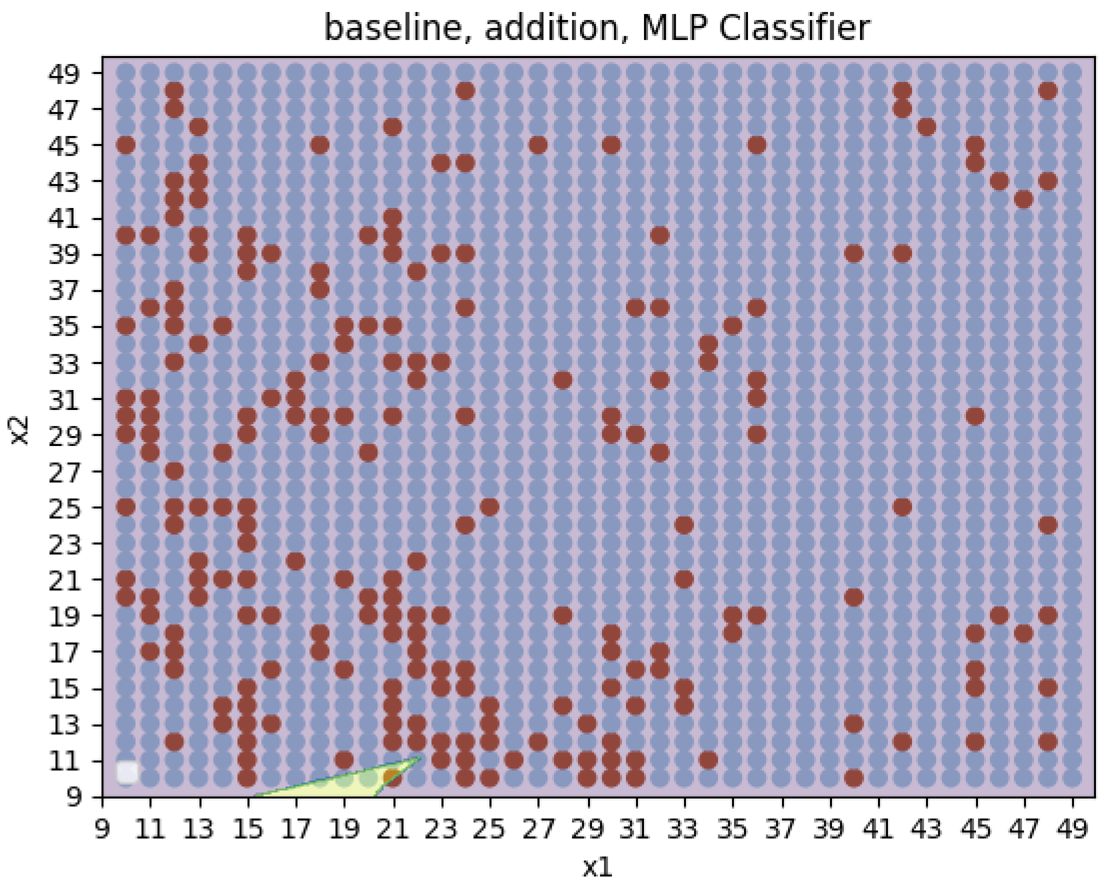
```
      strategy miss_rate (in %)  accuracy (in %)  classifier_accuracy (in %)
0  word_problem           100.0         5.437500                     94.5625
1          code           100.0         2.562500                     97.4375
2       algebra             N/A         0.000000                      0.0000
3    in_context       89.198606        17.937499                     82.8750
4      baseline       94.977169        13.687500                     85.9375
```

Classifier Plots




code, addition, MLP Classifier
x1
x2


in_context, addition, MLP Classifier
x1
x2


baseline, addition, MLP Classifier
x2
x1
9
11
13
15
17
19
21
23
25
27
29
31
33
35
37
39
41
43
45
47
49

AI Collaboration

I used ChatGPT for getting the regex function to get the first numerical value out of the output string. Also used chatGPT to get boilerplate code generating graphs and training a MLP classifier.

This is not an AI collaboration but general shameless code copying. I took how to run 4-bit quantized 7B alpaca-lora model on colab from
https://www.mlexpert.io/machine-learning/tutorials/alpaca-and-llama-inference
This model works surprisingly well and has a lot better results.

Extra Credit

I ran the facebook/opt-2.7b model on a multiplication task and have the following results.

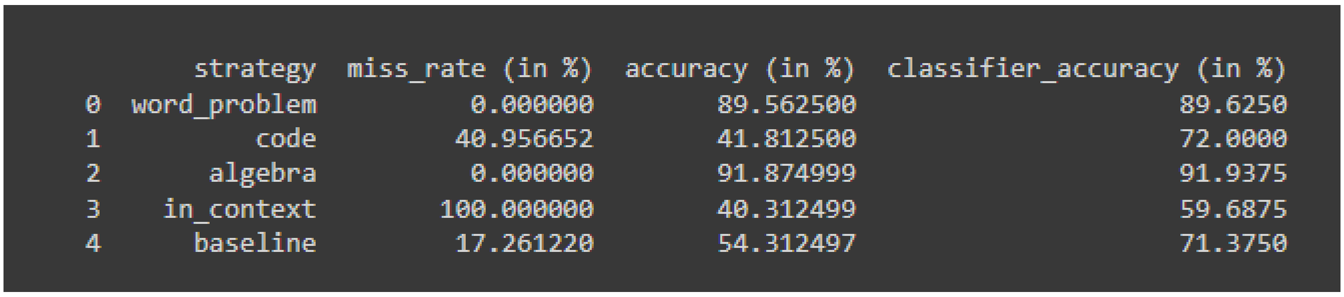
```
       strategy miss_rate (in %)  accuracy (in %)  classifier_accuracy (in %)
0  word_problem        12.603496        67.937499                     75.3125
1          code              N/A         0.000000                      0.0000
2       algebra              N/A         0.000000                      0.0000
3    in_context        97.222222         6.750000                     93.3125
4      baseline        75.886525         8.812500                     93.0625
```

The LLM did well on the word problems but for the remaining strategies had a poor performance.
I have attached the code and pdf with generated outputs if required.

—-------------------

I ran the addition task on the alpaca-lora 4 bit quantized model, the updated weights and model is available on hugging face.

Here are the results.



```
       strategy  miss_rate (in %)  accuracy (in %)  classifier_accuracy (in %)
0  word_problem          0.000000        89.562500                     89.6250
1          code         40.956652        41.812500                     72.0000
2       algebra          0.000000        91.874999                     91.9375
3    in_context        100.000000        40.312499                     59.6875
4      baseline         17.261220        54.312497                     71.3750
```

We can clearly see that the model performs very well with high LLM accuracy, High Classifier accuracy and low miss rate for some strategies.

Note : All the notebooks (facebook opt 2.7b, gpt neo 1.3b, multiplication and alpaca-lora model, and corresponding generated output PDFs have been attached with this.