# Introduction

In this project, our goal was to investigate whether and how transformer language models can aid with problem formulation by suggesting abstractions for problem statements.

We created a computational pipeline using the GPT-Neo transformer language model to reformulate a problem statement by submitting a bunch of training examples of similar nature, and evaluated model performance against human judgments. We also investigated how model performance varies by temperature (variability of results), number of parameters (model size), number of iterations, amount of training data, length of the result among other things.

# System

## Task formulation

The python function (script) is specifically developed for this problem and is tightly dependent on the structure of the input prompt. The text file has the input. You also need to keep a track of the number of input problem statements as it is required for parsing and generating output.

Optionally, you can set the argument in the function to 1 if you want the function to save the output in the csv. The function returns a pandas dataframe which you can then save to a csv.

One very important argument for the language models is temperature. It is like a measure of variability which ranges from 0 to 1. Higher the temperature, higher the randomness in the output. If the temperature is set to 0, the model doesn’t learn a lot and it just returns the problem statement as in. We did an experiment where we ranged temperature from 0.1 to 1 with step size of 0.2 from 0.1 to 0.5 and then a step size of 0.1 from 0.5 to 1. We found that the results were the best between 0.7 to 0.9.

In the interest of time and computing power, the results were generated and compared with a temperature of 0.9.

## Implementation

Libraries to be installed – numpy, pandas, random, string, datetime.datetime, time, transformers.pipeline

# Experiment methods

## Models tested:

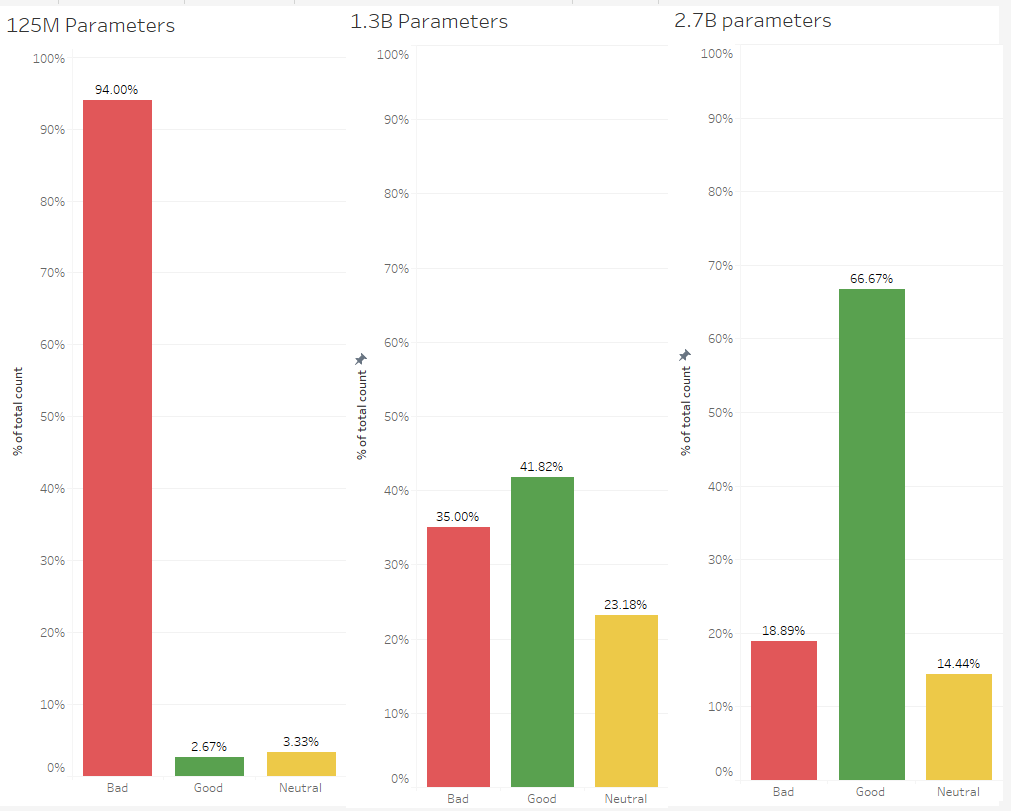
* GPT Neo with 125M Parameters
* GPT Neo with 1.3B Parameters
* GPT Neo with 2.7B Parameters

## Coding approach

The results of the models were manually coded based on human intuition.

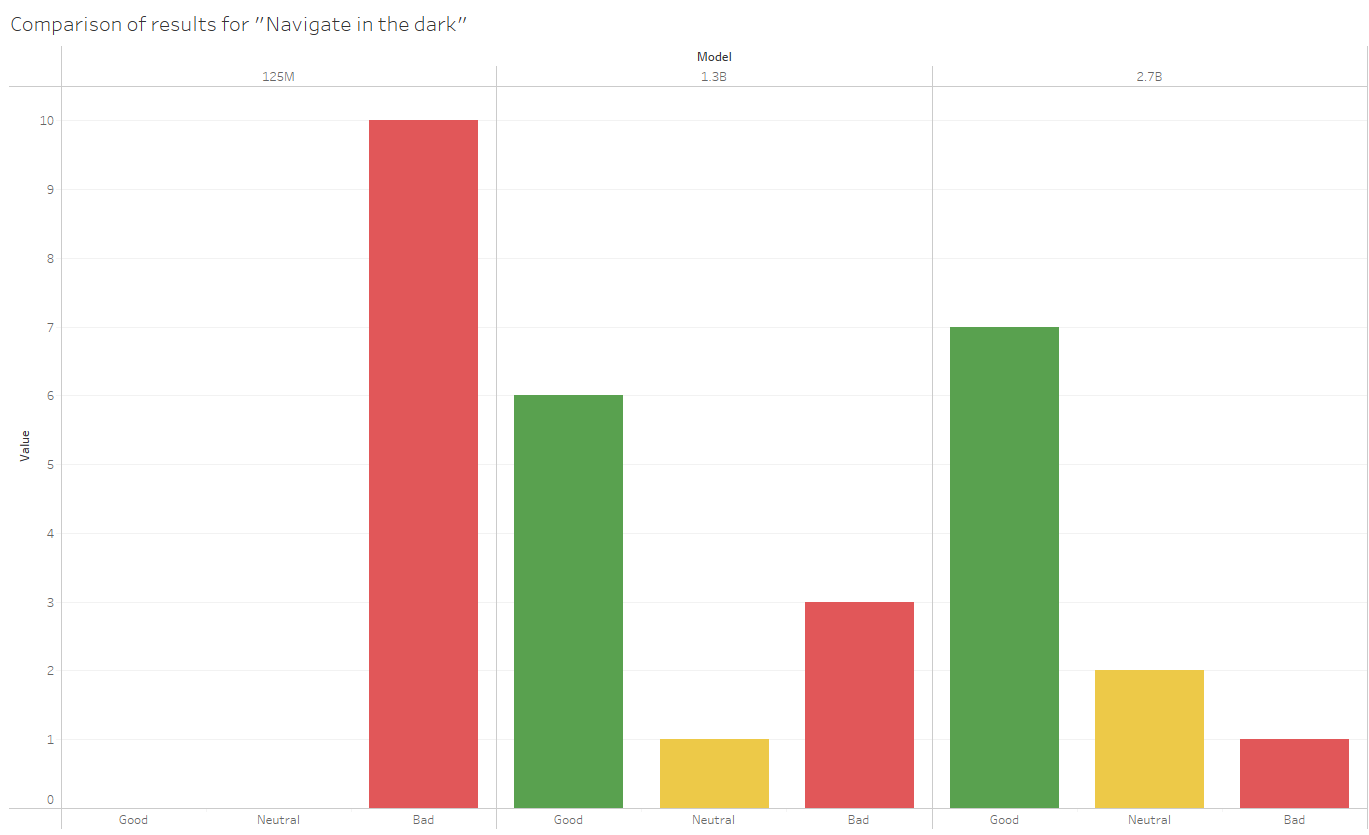
e.g., stop head from moving while sleeping on bus --> "prevent head motion while sleeping on bus" = bad; "prevent head motion while sleeping" = ok; "prevent unwanted motion on object" = good

# Results



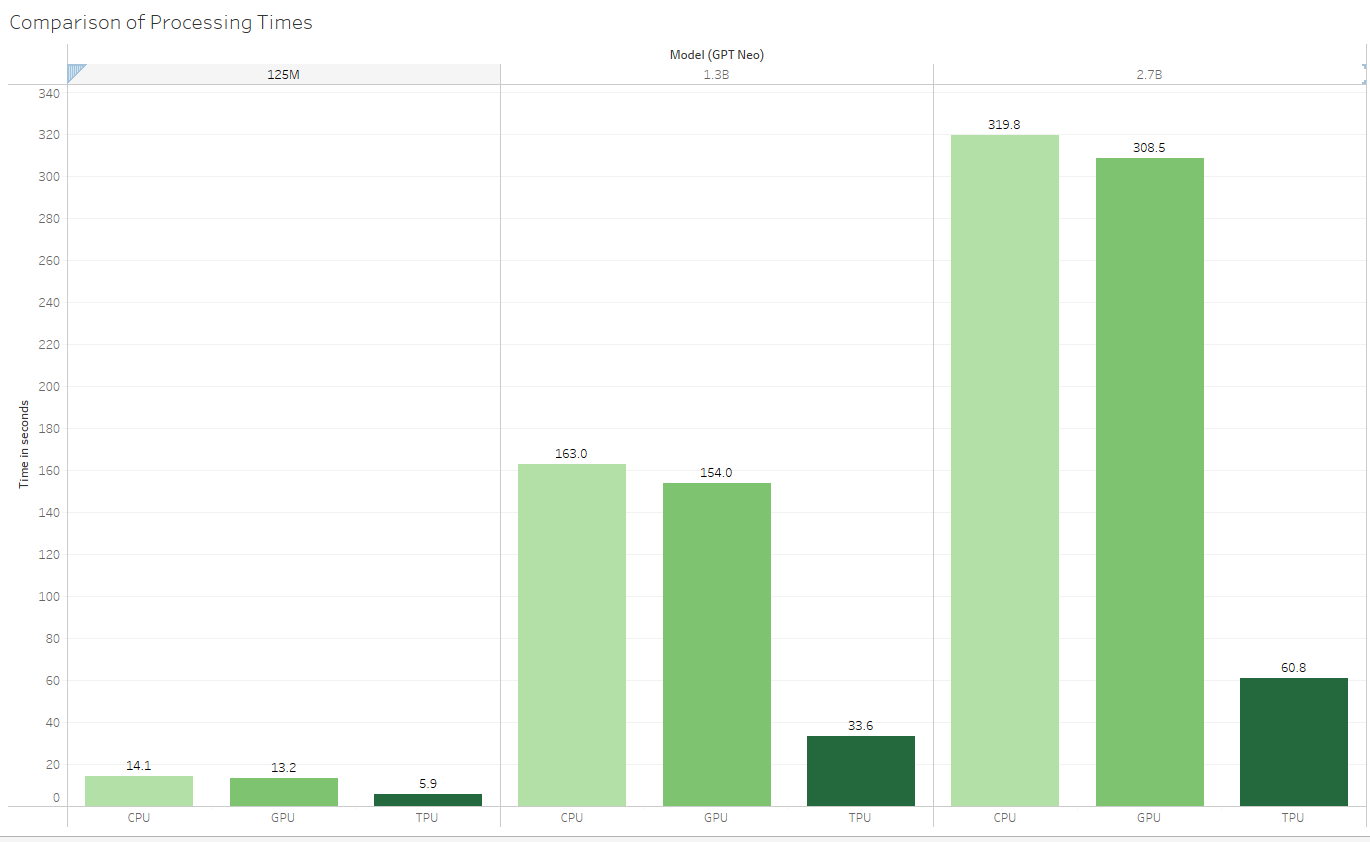
From the above diagram, we see that in general the model with 125M parameters generated extremely bad results with just 2.67% “good” outputs. Models with 1.3B and 2.7B parameters generated fairly good outputs with the 2.7B parameters model generating 66.67% “good” outputs followed by 41.87% for 1.3B model. All the models were tested on the same problem statements (around 15) with around 10 iterations for each.

For a problem statement “navigate in the dark”, following were the results:



|  |  |  |  |
| --- | --- | --- | --- |
| Model | Good | Bad | Neutral |
| 125M | 0 | 10 | 0 |
| 1.3B | 6 | 3 | 1 |
| 2.7B | 7 | 1 | 2 |

Comparison of average processing times per iteration for all the three models on CPU, GPU and TPU.



The processing times were calculated on Google Colab Pro. We see that processing times are significantly higher (almost double) as compared to the 1.3B model and almost 12 to 15 times as that of 125M model. Based on your priorities and preferences, you can choose the model and handle the tradeoffs.