# Experimental Design

## Objective

The objective of this study is to assess the programming ability of generative AI models. The current study focuses on solving programming challenges of various difficulty and type of leetcode.

Once the ability of one LLM has been evaluated along various benchmarks, the ability of other LLMs can be used for comparison.

## Evaluation

There are multiple ways of evaluating the coding ability. The first is the binary capability of the model to pass all test cases or not. Since the task is repeated multiple times, (up to five times) another benchmark is min, max or average solving capability. For example, the challenge might be solved correctly 3/5 times. This would result in a min = true, max = false and average = 3/5.

Another discrete measure is the number of test cases solved. 7/10 would result in a 70 % score for one test. Repetitions could be aggregated.

A more qualitative measure is the time and memory efficiency of the test cases solved. The test cases also provide the percentile of time and memory efficiency in comparison to the other user.

## Optimisation

Initially in the one shot approach, the models ability to answer the raw question is evaluated. However, there are multiple options to increase the capability of the model.

**Feedback loops**

In case not all test cases are passed, the model can receive the error message. This process can repeated X times and the program’s ability to take into account the user’s feedback can be evaluated.

**Solution structuring**

Another often mentioned optimization method is the structuring of the answer before coding the solution. In that case, the prompt contains the command to first provide a detailed structure of the problem before coding.

MORE TO FOLLOW

# Implementation

## One shot

Dataset: sampled\_df

Constraints: Only before 2021, easy, medium and hard questions from all types

Repetitions: 3 times (should get up to 5 times in my opinion)

Remarks: There are only 2 hard optimization problems (for free)

## Feedback loops

Dataset: sampled\_df\_hard

Constraints: Only before 2021, hard and medium (if necessary) questions, 2 feedbacks given (max 3 times coding the same solution before evaluation)

Repetitions: Hopefully 5 times

Remarks:

* Since we only have 2 hard questions under the constraints, 3 medium questions have been added
* There is actually one question that gets used twice (1402)

Suggestions on evaluation from chatgpt

BINARY

Sure, here are some suggestions on the types of graphs and statistics you might want to create:

1. \*\*Success Rate Over Attempts:\*\* Plot a line graph to show the success rate (ratio of successful attempts to total attempts) of the program over multiple attempts. This can be done for each type of challenge, or overall. This would give an insight into the learning curve of your program.

2. \*\*Success Rate by Challenge Type:\*\* A bar chart could be used to show the success rate for each type of challenge, like array, sorting etc. This will give you a clear picture of which challenge types your program handles best.

3. \*\*Revision Success Rate:\*\* Calculate the success rate of the program after revising its answer. This can be displayed as a line graph showing the success rate after each revision. This would provide insights into how well the program is able to process feedback and correct itself.

4. \*\*Boxplot of Attempts Needed Per Challenge Type:\*\* This can give you an insight on which challenge types tend to require more attempts to solve, indicating their difficulty level for your program.

5. \*\*Heatmap of Success Rate:\*\* A 2D heatmap can be used to show the success rate with challenge type on one axis and attempt number on the other. This could give a good overall view of how success rate varies with challenge type and attempt.

6. \*\*Improvement Over Time:\*\* If you have data over a long period of time, you can plot a line graph showing how the overall success rate of the program has improved over time.

7. \*\*Statistics:\*\* Basic statistics like mean, median, mode, and standard deviation for the number of attempts per challenge and success rate can provide a concise summary of your program's performance.

Remember, the choice of graphs and statistics will depend on the nature of your data and what insights you are looking to extract from it. It may take some experimentation and data exploration to find the most informative visualizations.

CONTINUOUS

Thanks for clarifying. Here are some additional suggestions that take into account the percentage of test cases passed:

1. \*\*Average Test Case Success Rate:\*\* For each challenge type and attempt number, compute the average percentage of test cases passed. This can be displayed as a bar chart or a line graph.

2. \*\*Heatmap of Test Case Success Rate:\*\* Similar to the previous heatmap suggestion, you could create a 2D heatmap with challenge type on one axis and attempt number on the other. The color at each point in the grid would represent the average percentage of test cases passed for that combination of challenge type and attempt number.

3. \*\*Progression of Test Case Success Rate:\*\* Plot a line graph showing how the percentage of test cases passed changes over the attempts for a particular type of challenge. This will give an idea of how the program improves over attempts.

4. \*\*Distribution of Test Case Success Rate:\*\* Plot histograms or boxplots for the distribution of the percentage of test cases passed for each type of challenge. This can help identify any skewness in the results and pinpoint challenges that are particularly difficult or easy for your program.

5. \*\*Correlation between Attempts and Test Case Success Rate:\*\* You can create a scatter plot with attempt number on one axis and percentage of test cases passed on the other. This can help identify trends or patterns between the number of attempts and success rate. You can compute the correlation coefficient to quantify this relationship.

6. \*\*Statistics of Test Case Success Rate:\*\* Similar to basic statistics on the number of attempts, you can compute mean, median, mode, and standard deviation for the percentage of test cases passed. This can provide insights into the central tendency and variability of your program's performance.

7. \*\*Revision Impact on Test Case Success Rate:\*\* Plot a line graph showing how the percentage of test cases passed changes after each revision. This would provide insights into how effectively the program uses the feedback to improve its answers.

What I am going to measure:

* Line plot showing all walks of not perfectly solved attempts
* Percentage of unsolved cases in first attempt solved by repetition
* Average increase of test cases passed from attempt 0 to 1/ attempt 1 to 2
* Average increase of test cases passed per attempt by
  + Starting with cases passed at attempt 0 below 50 %
  + Starting with cases passed at attempt 0 above 50 %

## Memory and runtime optimization

Tell the program to achieve minimal runtime and memory. See how it reacts.

## Giving a role?

You are a pro programmer …

## Wisdom of crowds

What if I put multiple agents in parallel and let one decide what answer to take

## System message engineering

Hallucinated outputs in function calls can often be mitigated with a system message. For example, if you find that a model is generating function calls with functions that weren't provided to it, try using a system message that says: "Only use the functions you have been provided with."