# 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.

Say that initially no system variables were used. Give example of EXP01 golang\_no\_sys result. (the constraints of keeping in alignment with the coding templates and not doing imports or main was only mentioned at the end of the prompt, not in the system variables) Then compare to the one with system variables used.

But even with the system variables in place, the system sometimes ignores them (creation of main in script although clearly instructed not to):

E.G. EXP01 golang\_sys – question 2267 attempt2

func main() {

m := 4

n := 6

guards := [][]int{{0,0},{1,1},{2,3}}

walls := [][]int{{0,1},{2,2},{1,4}}

result := countUnguarded(m, n, guards, walls)

fmt.Println(result)

}

**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 - 1

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

## Wisdom of crowds – 2 (with feedback)

Let them compare their results and see if they can come up with suggestions of an improved solution that the final agent codes down

## 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."

## Testate question vs. feedback feeding – what is the chance in each case that it results in a correct code solution

## Question types

1. **Algorithmic Thinking and Problem Solving**: All the categories would fit into this dimension, but Dynamic Programming, Sorting, and Simulation problems especially emphasize the ability to understand and design algorithms, and to break down complex problems into smaller, manageable parts.
2. **Data Structure Understanding and Utilization**: Hash Table, Sorting, and String problems strongly test the understanding of data structures and their usage in various contexts.
3. **Mathematical Skills**: Math problems, obviously, fall into this category. But also, Dynamic Programming often requires mathematical thinking, particularly when it comes to understanding recursion and optimization. Similarly, certain Sorting problems could also require a basic understanding of math concepts.
4. **Software Engineering and Design Principles**: Design problems focus on this dimension, testing your ability to architect systems, apply design principles and patterns, and make good design decisions based on requirements and constraints.
5. **Implementation and Coding Skills**: All categories test this skill, but String problems often require careful attention to detail in the implementation, as small mistakes can easily lead to bugs. Simulation problems also tend to involve complex coding tasks as you implement the rules of the simulation.
6. **Optimization Techniques**: Dynamic Programming and Sorting problems often require optimizing solutions for better efficiency, either in terms of time or space complexity. Hash Table problems can also fall into this category, as hash tables are often used for optimization purposes.
7. **Understanding of Real-world Applications and Systems**: Design, Simulation, and Hash Table problems often require or develop an understanding of real-world systems, whether it's designing a real-world system, simulating a real-world process, or using a hash table to solve a problem that arises in practical applications.

## Compare programming languages

### Languages

java

python3

javascript

csharp

golang

ruby

**other languages out of scope for now**

kotlin, mysql, php, python, , ,rust, scala, swift, typescript

### EXP01

First take a sample of 40 questions of medium difficulty (5 from each category). Then go into detail for each category. (3 attempts), no feedback. All questions come from after 2021.

python3

Proxies for identifying availability of documentation and examples

1. **Search engine results**: Perform a search for "[programming language] documentation" or similar phrases in a search engine like Google and see how many results you get. This is a very rough approximation, as the number of results also depends on the search engine's indexing and does not necessarily reflect the amount of relevant or high-quality content.
2. **Official documentation**: Evaluate the extent of the official documentation. Larger, more comprehensive official documentation suggests a more well-documented language.
3. **GitHub repositories**: Look at the number of repositories related to each language on GitHub. This gives a sense of how much open-source code there is, which often includes documentation. Additionally, GitHub's "Used by" feature can give you a sense of how widely used a language is in the community, which often correlates with documentation volume.
4. **Stack Overflow questions**: The number of questions tagged with a certain programming language on Stack Overflow can be an indicator of the volume of unofficial, community-provided documentation. More questions and answers mean more problems are being discussed and documented.
5. **Tutorials and courses**: Count the number of online courses and tutorials available on platforms like Coursera, Udemy, Codecademy, and freeCodeCamp. More courses suggest more documentation.
6. **Books**: Look at the number of books published about the language. Websites like Amazon could provide this data.

## Command that it comes up with test cases