Comparative Performance Analysis of Vanilla and Island Model Genetic Programming in Symbolic Regression

Muhammed Bilal
Department of Computer Science
Brock University
St. Catharines, Canada
bb18hb@brocku.ca

Hamza Sidat
Department of Computer Science
Brock University
St. Catharines, Canada
hs18so@brocku.ca

***Abstract*—This research paper presents a demonstration and comparison of Symbolic Regression using Vanilla (standard) Genetic Programming with Island Model Genetic Programming, implemented using the DEAP library. Symbolic regression involves the automatic discovery of mathematical expressions that best fit given datasets. The paper explores the application of genetic programming techniques to evolve symbolic expressions that predict target functions represented by sample dataset text files**

I. Introduction

Genetic Programming (GP) is an evolutionary algorithm with the objective to seek solutions in cases where traditional methods fail. Symbolic regression has a number of important applications, ranging from trying to find the best model fit for a given set of data to the generation and evolution of mathematical expressions. This method gives a flexible way of model identification and proves very crucial in different fields, ranging from data science to engineering [1].

Our research project aims to show a comparative analysis of two well-known GP models: the Vanilla GP model and the Island Model GP. We therefore compare the two models, that is, the single-population traditional approach and the distributed evolutionary strategy, towards differences in solving performance differences, especially when solving symbolic regression problems. Knowing the strengths and weaknesses of each model gives us the possibility of choosing the best method for our problem and could also, along with the knowledge of problems to address and adapt, lead to better ways of using Genetic Programming.

The vanilla GP, also referred to as Standard GP, is a simple evolutionary algorithm in which the candidate solution population evolves through generations by means of genetic operations of crossover, mutation, and selection. This model has laid the foundation for the development and application of GP due to its simplicity and large areas of applicability [2].

On the other hand, Island Model GP is complex in that it first places candidate populations on separate ”islands,” where they will evolve independently for a number of generations before migrating individuals between islands take place. The model is thought to increase diversity in possible solutions and has shown promise for accelerating results and improving quality in the solutions of different scenarios [3].

Despite the fact that their uses are extensive, there does exist one comparison that should most definitely be conducted systematically, rigorously, and under controlled conditions. The comparison which certainly needs to be done is in the domain of symbolic regression. Therefore, such studies are relevant in the way that they address performance metrics not in terms of accuracy, computational efficiency, and solution robustness, but rather give deep insight into how these models perform in an array of different types of regression problems. In this project, we tried to present a comprehensive analysis that may be beneficial in improving the GP techniques and deriving more effective symbolic regression algorithms.

In this paper, we explore the characteristics of these models in more depth, assess their performance on standard symbolic regression tasks, and offer views that might influence future applications and developments in genetic programming.

II. Experimental Setup

A. Overview of Genetic Programming Models

1) Vanilla Model

Vanilla Genetic Programming is a very basic form of GP based on the most primitive methods of natural selection and genetic evolution. It uses a very simple evolutionary algorithm, where a single population of candidate solutions (computer programs, mathematical expressions) is evolved over generations.

A Vanilla model of GP consists of populations and generations, where a population is involved in the inheritance of generations of the evolutionary process, genetic operators, such as crossover (recombination), mutation, and selection (choosing the fittest individuals), that direct the evolution. The representation of solutions in GP is in the form of tree structures. Lastly, the fitness evaluation is used for the selection of individuals. Fitness evaluation is applied to each individual, which is intended to find out the quality of how well the symbolic regression (in our experiment) is solved by that individual [2].

2) Island Model

The Island Model GP extends the basic principles of GP by distributing the population across multiple subpopulations (often called 'islands' or 'demes'), each evolving independently with occasional migrations of individuals between islands.

Key features of the Island model of GP consist of the distributed population that concurrently evolves and comprises of few separate, distributed, and independent populations that explore different areas of the solution space. Secondly, the migration mechanism involves the migration of some chosen individuals from one island (deme) to another, which occurs over time intervals. This migration tends to increase diversity. Thirdly, parallel computation is an important feature that speeds up the processing time of the evolutionary algorithm. Lastly, Island-independent parameters adaptation, that is, different islands (demes) may have their own set of GP parameters, such as mutation rate, selection pressure, etc), which makes the evolutionary process more diverse [3].

B. Parameters Listed

1) GP Parameters

In this section, we present a comprehensive overview of the genetic programming (GP) parameters used in our tests for both Vanilla and Island Model GPs. These parameters, while shared, may behave differently in each model. They play a vital role in guiding the actions of the evolutionary algorithm and significantly impact the quality of the solutions achieved in each model.

Meanings and Explanations of parameters:

- Crossover: Crossover facilitates exploration by generating diversity in the population. It allows for the exchange of genetic material between individuals, potentially creating offspring with beneficial combinations of traits that were not present in the parent solutions. This diversity helps prevent premature convergence to sub-optimal solutions by exploring different regions of the search space.
- Deme Size: Deme size refers to the amount of individuals in each subpopulation (Deme) within the Island model. Larger Deme size can increase the diversity, but also lead to a slower convergence.
- Elitism: Elitism promotes exploitation by preserving the best solutions throughout the evolutionary process. By ensuring that the best individuals survive from one generation to the next, elitism helps maintain or improve the overall quality of solutions over time by preserving some of the fittest individuals.
- Migration Interval: Migration interval determines how often migration between demes occurs, measured in terms of generations. Through our Island model in Table II, migration between demes happens every 5 generations. Migration exchanges individuals between demes, promoting diversity across demes and helping in sharing beneficial genetic.
- Mutation: Mutations adds an element of randomness to the individuals, utilizing a bit of exploration as well as exploitation to the problem. It achieves this by introducing a bit of new genetic material to the individual, potentially causing unexpected results. Excessive mutation, however, can also hinder the convergence of the evolutionary process.
- Number of Demes: This parameter refers to the number of islands or subpoplations in the evolutionary process. This architecture allows for parallel evolution, increasing exploration of different regions of the search space.

TABLE I
DEFAULT VANILLA GP PARAMETERS

Parameters	Values
Population Size	500
Crossover Probability	90%
Mutation Probability	10%
Generations	75
Number of Elites	1
Tournament Size	3
Min Tree Size (Init.)	3
Max Tree Size (Init)	7
Max Tree Size	17
# of Runs	10

TABLE II
DEFAULT ISLAND MODEL GP PARAMETERS

Parameters	Values
Deme Size	500
Number of Demes	4
Migration Interval	5
Migration Rate	10%
Crossover Probability	90%
Mutation Probability	10%
Generations	75
Number of Elites	1
Tournament Size	3
Min Tree Size (Init.)	3
Max Tree Size (Init)	7
Max Tree Size	17
# of Runs	10

2) Training/Testing Data

- Size of Training/Testing Data:
  - We divided our dataset into two subsets: a training set and a testing set. The training set comprises 70% of the total data, while the testing set contains the remaining 30%. This composition was chosen so that there is an adequate amount of training being done on the evolutionary model for both Vanilla and Island, while also reserving a portion for testing each model's performance on unseen data.
- Method of Selecting Training/Testing Data:
  - Simple Train/Test Split: We decided to go for a simpler approach where the dataset is shuffled randomly to ensure that the training and testing sets have a similar distribution of data points as well as to avoid introducing bias into the dataset split. This simplicity is because our dataset is not immensely complex with specific characteristics or patterns, such as dependencies.

C. Dataset Selection

We selected our dataset by generating the data using mathematical formulas in Microsoft Excel. The two formulas we used were:

$$f(x) = (x^4 + x^3 + x^2 + x)^2 \tag{1}$$

and

$$f(x) = \sin(x^2) + \frac{1}{2}\cos(x) + \frac{1}{3}\sin(2x) \tag{2}$$

We generated 100 rows of data using these formulas and split it accordingly.

D. GP Language and Operators

TABLE III
VANILLA AND ISLAND MODEL FUNCTION SET

Function	Arity	Example
ADD	2	x + y
SUB	2	x - y
MUL	2	$x \times y$
DIV	2	$\frac{x}{y}$ (1 if $y = 0$)
NEG	1	-x
SIN	1	$\sin(x)$
COS	1	$\cos(x)$

TABLE IV
VANILLA AND ISLAND MODEL TERMINAL SET

Terminal	Action
EPH	rand101 (-1 to 1)
x	(Renamed from ARG0 to 'x')

Meanings and Explanations of GP Language:

- ADD: This function performs the addition operation on two numbers.
- SUB: This function performs the subtraction operation on two numbers.
- MUL: This function performs the multiplication operation on two numbers.
- DIV: This function performs the division operation on two numbers. Automatically returns the result as 1 if the denominator is equal to 0.
- NEG: This function performs the negation operation on a number.
- EPH: This terminal generates a random number between -1 and 1 inclusive, which persists throughout the entire evolutionary run.

E. Fitness Formula

The fitness function serves as the guidance of the evolutionary algorithm towards the goal, which in our case is to represent the original model through symbolic regression in both Vanilla and Island models. The algorithm from DEAP does this in a very simple way:

Here, a number of things are occurring: We are transforming the tree expression in a callable function. The points parameter represents the data points used for evaluation. Next, the mean squared error between the expression is evaluated between func(x) and the real function: x^4 + x^3 + x^2 + x. The formula (1) mentioned below is utilized in both models as a method of performance evaluation. In vanilla GP, each individual receives a certain fitness value and that value is seen to improve throughout the evolution process. In the Island model, since the population is divided based on the number of islands, each island consists of its own individuals with their own fitness values. Hence, we then take the average of the output values from the same generation of each island and then utilize this formula to calculate its fitness value.

1) Formula:

The fitness formula can be represented as:

$$\text{Fitness(ind)} = \frac{1}{N}\sum_{i=1}^{N}\left(f_{\text{ind}}(x_i) - (x_i^4 + x_i^3 + x_i^2 + x_i)^2\right) \tag{3}$$

F. Computational Environment

1) Software

- DEAP, or Distributed Evolutionary Algorithms, is a powerful Python-based open-source library that allows you to create evolutionary algorithms with a high degree of flexibility. It allows you to define your own operators,

types, and evaluation functions, making it very adaptable to different problem sets.

2) GP Models

- Vanilla Model: The basic form of Genetic Programming. It involves a population of individuals that evolve over time. The processes include selection, crossover (mating), and mutation operations. The fittest individuals, determined by a fitness function (evaluation), are selected for creating the next generation.
- Island Model: This is a more complex architecture of GP. Here, the population is divided into sub-populations known as Demes (islands). Each Deme independently evolves its individuals. Occasionally, individuals migrate from one island to another. This model promotes diversity and can help prevent premature convergence to sub-optimal solutions.

3) Performance Metrics

- Convergence Quality: Depends on the analysis of multiple factors, such as fitness progression, stagnation, and problem complexity.
- Computational Time: The time each model's evolutionary process takes.

III. Experiments

In this section, a series of experiments are performed to compare the two models, Vanilla or Standard GP and Island Model GP. These experiments evaluate the two models in various scenarios considering factors such as convergence performance as well as computation time.

TABLE V
Experimental Factors

Factor	Description
Convergence	Performance of convergence for Vanilla vs. Island
Computational Time	Computational time for Vanilla vs. Island

TABLE VI
Experiments

Experiment	Description
1	Vanilla Vs Island with simple data-set (1)
2	Vanilla Vs Island with complex data-set (2)

IV. Results

A. Experiment 1 - using simple dataset (1)

1) Performance Graphs

Vanilla Model with simple dataset:

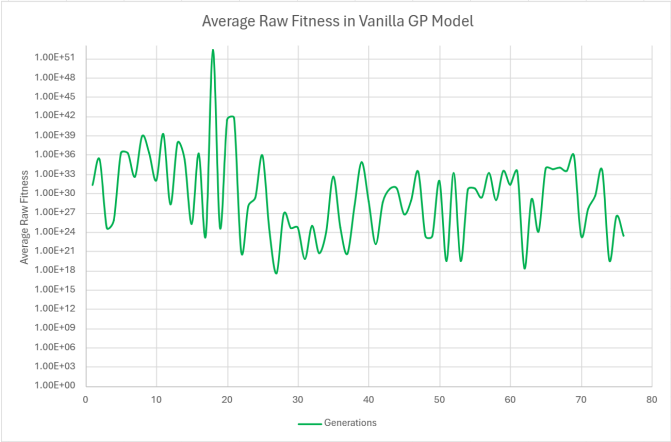

Fig. 1. Average of average Fitnesses using Vanilla Model with simple dataset and default GP parameters

Factors:

- Convergence: the convergence of the graph in Fig. 1 seems to be really poor at first glance, but when you minimize the outliers, we can then say that convergence is demonstrated, causing a general pattern of fitness values going downwards and decreasing. Note that these values fluctuate a lot because they are raw fitness values and raw fitnesses can fluctuate a lot depending on randomness in the processes with the parameters. (crossover, mutation, etc.) Overall, due to constant fluctuations, the convergence of the fitness is not as bad as it looks but is not anything spectacular either.
- Computational time: The computation time for the vanilla GP model for symbolic regression took 886.64 seconds or approximately 15 minutes to compute 10 runs, each with a solution.

Island Model with simple dataset:

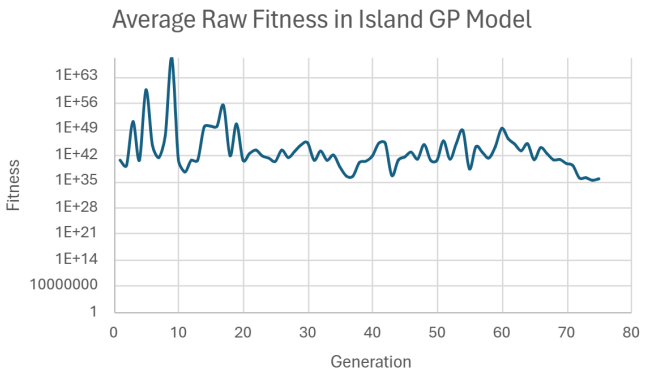

Fig. 2. Average of average Fitnesses using Island Model with simple dataset and default GP parameters

Factors:

- Convergence: the convergence of the fitness graph in Fig. 2 is converging poorly for the first 10 to 30 generations, and after that, the convergence is seen. The last few generations can be seen converging in a consistent manner.
- Computational time: The computation time for the Island GP model for symbolic regression took 2383.56 seconds or approximately 40 minutes to compute 10 runs, each with a solution being evaluated on the testing dataset.

2) Summary Tables

TABLE VII
SUMMARY OF COMPUTATIONAL TIME AND CONVERGENCE

Model	Computational Time (minutes)	Convergence (performa
Vanilla GP	14.78	poor
Island Model	39.726	poor

3) Training and Testing

Vanilla Model: Out of the 10 runs, the individual with the best fitness was chosen to be executed on the remaining 30% of the dataset, also known as the testing data. This is to determine whether our evolutionary process in the vanilla model was adequate to produce a solution or individual for accurately modeling our dataset. Our solution performed beautifully with an error rate of 1.133667926462279, which represents its raw fitness value. When it comes to symbolic regression, we want to minimize the error between the modeled function and the actual function. Hence, the lower the raw fitness value, the better the individual is, making this solution very accurate at modeling the dataset. So even though the Vanilla model, overall did not perform too well in training, it ended up coming up with a great solution.

Island Model: After performing 10 runs on the training dataset, which consisted of 70% of the dataset, we selected the best individual among these 10 runs and executed this individual on our testing dataset. This solution performed comparatively poorly than Vanilla Model's best solution with an error rate of 19968775724.7837, that is, also the raw fitness of this individual on the testing dataset. This result was surprising because the Vanilla model performed poorly on the average fitness plot, but its best individual performed significantly better than the Island model.

4) Results

Difference: Overall, through the plotted fitness graphs shown in Fig. 1 and Fig. 2, it is evident that the convergence of the Island Model is much smoother and consists of fewer fluctuations in its fitness values. However, the better and fitter solutions are still found in the Vanilla model than in the Island model. Therefore, whether to say which model is better is a tough decision, but due to its large computational time, almost poor convergence, and lack of good solutions because of poor performance on testing data, one can argue that the more efficient model for simple numbered datasets is the vanilla GP model.

B. Experiment 2 - using complex trig. dataset (2)

1) Performance Graphs

Vanilla Model with the complex dataset:

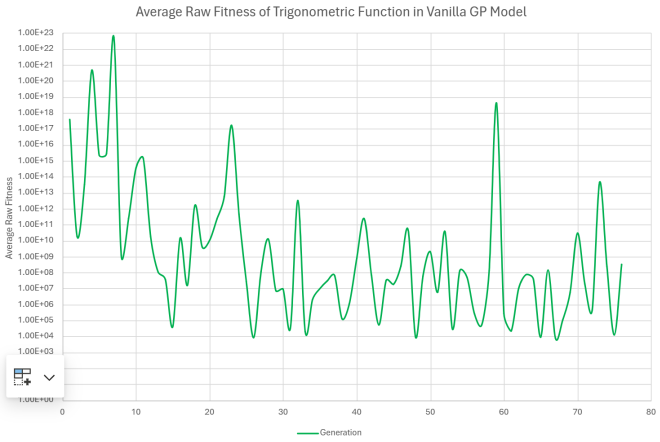

Fig. 3. Average of average Fitness using Vanilla Model with complex trig. dataset and default GP parameters

Factors:

- Convergence: Despite some fluctuations in the average fitness values, the convergence of the graph in Fig. 1 seems to be following a pattern of steadily decreasing fitness values, resulting in fitter individuals as generations pass. One thing to note is that the log of the vertical axis had to be taken to reduce the complexity of the large fitness values.
- Computational time: The computation time for the vanilla GP model for symbolic regression took 407.77 seconds or approximately 7 minutes to compute 10 runs, each with a solution.

Island Model with complex dataset:

Factors:

- Convergence: As we can see in Fig. 4, which is the plot of the average fitness of ten runs on a dataset that included trigonometric functions. We can see the convergence in the plot, despite the complexity of the data. Secondly, we also noticed the difference in the performance of the island model on a simple dataset and the dataset of trigonometric functions.

- Computational time: The computation time for the Island GP model for symbolic regression on the trigonometric

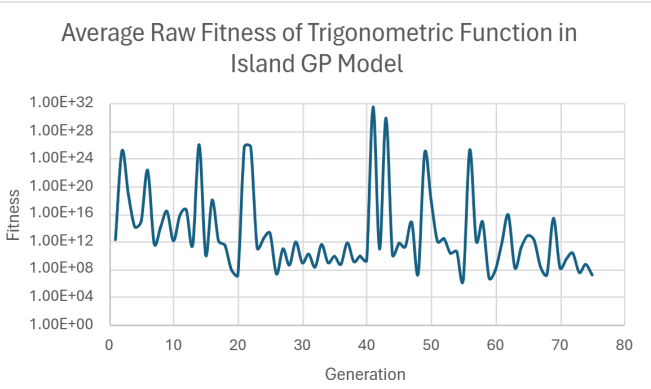

Fig. 4. Average of average Fitness using Island Model with complex trig. dataset and default GP parameters

functions dataset took 706.92 seconds or approximately 12 minutes to compute 10 runs, each with a solution being evaluated on the testing dataset. This finding was also interesting, because despite adding two primitives (sin and cos)), the computational time was relatively lesser than a simple dataset.

TABLE VIII
SUMMARY OF COMPUTATIONAL TIME AND CONVERGENCE

Model	Computational Time (minutes)	Convergence (performance)
Vanilla GP	6.7962	average
Island Model	11.782	average

2) Training and Testing

Vanilla Model: Out of the 10 runs, the individual with the best fitness was chosen to be executed on the remaining 30% of the dataset, also known as the testing data. This is to determine whether our evolutionary process in the vanilla model was adequate to produce a solution or individual for accurately modeling our dataset. Our solution performed very accurately with an error rate of 0.013938612971503232, which represents its raw fitness value. When it comes to symbolic regression, we want to minimize the error between the modeled function and the actual function. Hence, the lower the raw fitness value, the better the individual is, making this solution very accurate at modeling the dataset. So even though the Vanilla model, overall did not perform too well in training, it ended up coming up with a great solution when it came to detecting patterns on unseen data.

Island Model: After training our island model on the training dataset of trigonometric function, we selected the best individual among these 10 runs and executed this individual on our testing dataset. This solution performed comparatively poorly than Vanilla Model's best solution with an error rate of 0.000698363963757725, that is, also the raw fitness of this individual on the testing dataset. The rate of error, which is the difference between an actual solution and a GP solution is significantly small, resulting in relatively better performance than the Vanilla model.

3) Results

The results achieved through the symbolic regression of the complex trigonometric dataset proved to be surprising. Trigonometric functions are known to be immensely complex with values constantly fluctuating and oscillating between intervals. This pattern makes them a challenge to model. Furthermore, we decided to utilize the function sets, "sin" and "cos", which might have aided in the modeling of the trigonometric functions. The addition of these functions helped detect different types of periodic patterns, therefore improving the fitness of the individuals.

Difference: Overall, through the plotted fitness graphs shown in Fig. 3 and Fig. 4, it can be seen that the Vanilla model generates a somewhat smoother and downward convergence curve, whereas the Island model consists of more ups and downs with less of a downward curve, showing that most fitness values do not decrease as much. However, we should note that the best solution in the Island model outperforms the best solution in the Vanilla model. From this result, it can be deduced as the opposite of the first experiment consisting of the simple dataset (1). After comparing the time difference and analyzing the best solutions from both models, it is stated that the Island model is more efficient at finding a better solution for modeling more complex trig functions since the computational time difference between both models is not drastic with the Island model evolving a much better solution.

V. CONCLUSION

Through our experiments above, we observed that when it comes to symbolic regression, modeling simple datasets is more suited to and efficient using Vanilla GP architecture. On the other hand, modeling much more complex datasets, such as datasets containing trigonometric functions, is easier to compute utilizing the Island model architecture. The key to the Island model lies within its ability to maintain diversity in the population, which is very useful for complex functions. Complex functions, like trig functions, consist of multiple local minima, where getting stuck there is often easy when using Vanilla GP. The Island model, consisting of handling sub populations on multiple islands (demes), helps in increasing the search space and thereby, escaping sub-optimal solutions that the Vanilla model would usually get stuck at.

REFERENCES

[1] D. Angelis, F. Sofos, and T. E. Karakasidis, "Artificial Intelligence in Physical Sciences: Symbolic Regression Trends and Perspectives," *Archives of Computational Methods in Engineering*, vol.

30, pp. 3845–3865, 2023. [Online]. Available: https://doi.org/10.1007/s11831-023-09922-z

[2] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.

[3] Z. Skolicki, "An Analysis of Island Models in Evolutionary Computation," *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2005)*, Washington, DC, USA, pp. 1–8, ACM, 2005.

[4] F.-A. Fortin, F.-M. De Rainville, M.-A. Gardner, M. Parizeau, and C. Gagné, "DEAP: Evolutionary Algorithms Made Easy," *Journal of Machine Learning Research*, vol. 13, pp. 2171–2175, July 2012. Note: Open source software available at http://deap.gel.ulaval.ca

APPENDIX

A. Experiment 1

1) Vanilla Model Solution Tree Code:

sub(sub(sub(mul(sub(neg(0), protectedDiv(protectedDiv(mul(sub(protectedDiv(add(x, x), protectedDiv(-1, x)), sub(mul(add(sub(1, x), mul(-1, 0)), add(neg(x), sub(1, 0))), sub(add(neg(x), add(x, x)), add(protectedDiv(x, 1), add(x, -1))))), sub(add(sub(neg(neg(x)), mul(neg(x), add(0, 1))), sub(sub(sub(1, 0), sub(x, x)), protectedDiv(sub(x, 0), add(x, 0)))), mul(mul(sub(add(x, 1), neg(-1)), mul(protectedDiv(x, 1), sub(x, x))), add(0, neg(mul(x, 1))))))), add(-1, mul(add(sub(mul(protectedDiv(0, 0), -1), neg(mul(x, -1))), neg(mul(mul(add(sub(neg(neg(x)), protectedDiv(x, x)), sub(sub(sub(1, 0), sub(x, x)), protectedDiv(sub(x, 0), neg(x)))), x), mul(x, x)))), add(protectedDiv(protectedDiv(sub(x, x), neg(1)), mul(add(-1, x), protectedDiv(x, -1))), protectedDiv(add(protectedDiv(x, -1), neg(x)), add(mul(x, x), add(x, protectedDiv(x, -1))))))), x)), neg(neg(x))), add(neg(mul(add(x, x), add(-1, x))), add(mul(neg(mul(sub(x, sub(1, 0)), mul(mul(neg(-1), -1), protectedDiv(x, -1)))), x), mul(sub(sub(neg(protectedDiv(mul(protectedDiv(mul(add(sub(x, x), mul(x, -1)), sub(protectedDiv(protectedDiv(add(x, x), protectedDiv(-1, x)), x), x)), neg(add(neg(0), neg(x)))), protectedDiv(mul(mul(add(x, x), add(x, add(x, x))), mul(protectedDiv(1, x), protectedDiv(1, 1))), sub(protectedDiv(x, mul(x, x)), neg(neg(x))))), neg(x))), 1), 1), protectedDiv(x, x))))), mul(protectedDiv(add(x, x), protectedDiv(-1, x)), x)), add(neg(add(sub(1, 0), add(neg(x), add(x, x)))), add(sub(0, x), mul(-1, 0))))

2) Island Model Solution Tree Code:

mul(sub(neg(-1), add(x, 0)), sub(add(sub(neg(protectedDiv(sub(x, x), sub(0, add(sub(mul(neg(add(x, x)), -1), mul(sub(sub(x, 1), mul(x, 1)), 0)), sub(neg(add(mul(x, x), 1)), sub(mul(protectedDiv(x, x), sub(x, 0)), neg(neg(0))))))), neg(0)), protectedDiv(protectedDiv(mul(add(neg(add(sub(1, x), sub(0, -1))), sub(add(protectedDiv(0, 1), neg(x)), protectedDiv(add(1, x), add(0, x)))), sub(neg(0), add(add(mul(add(sub(mul(neg(sub(protectedDiv(neg(0), sub(1, x)), neg(0))), add(add(mul(protectedDiv(x, x), neg(x)), neg(add(0, x))), neg(add(neg(x), neg(x))))), add(mul(protectedDiv(mul(mul(1, x), add(x, x)), x), protectedDiv(protectedDiv(sub(-1, x), sub(1, x)), protectedDiv(add(x, x), add(x, x)))), neg(neg(sub(add(x, x), neg(x))))), add(mul(mul(x, sub(neg(sub(x, -1)), mul(mul(x, -1), protectedDiv(0, x)))), neg(mul(add(sub(-1, 1), 0), mul(sub(x, x), add(1, x))))), sub(protectedDiv(x, sub(mul(add(x, -1), protectedDiv(x, -1)), sub(add(x, x), neg(-1)))), sub(neg(protectedDiv(sub(x, x), sub(0, 1))), neg(1))))), neg(protectedDiv(0, mul(x, 0)))), x), x))), add(mul(protectedDiv(protectedDiv(sub(add(sub(mul(sub(x, -1), add(x, x)), protectedDiv(x, 0)), x), mul(mul(x, x), add(neg(x), mul(-1, -1)))), x), add(x, x)), neg(protectedDiv(0, mul(x, 0)))), mul(sub(0, protectedDiv(x, 0)), sub(neg(0), add(-1, x))))), add(protectedDiv(x, sub(add(x, x), sub(x, x))), protectedDiv(neg(protectedDiv(0, mul(neg(0), 0))), x)))), mul(mul(sub(mul(x, x), sub(add(x, x), x)), sub(add(x, x), x)), add(mul(mul(protectedDiv(1, 0), x), protectedDiv(-1, add(-1, 0))), mul(protectedDiv(x, x), neg(0))))))

B. Experiment 2

1) Vanilla Model Solution Tree Code:

sub(add(sin(mul(x, x)), cos(sub(0, mul(cos(neg(add(cos(x), cos(sub(neg(sin(neg(protectedDiv(protectedDiv(x, 1), sub(x, -1))))), protectedDiv(mul(neg(protectedDiv(protectedDiv(sin(0), 0), protectedDiv(-1, sin(x)))), neg(0)), sub(sub(sin(x), x), protectedDiv(cos(neg(mul(1, 1))), cos(neg(protectedDiv(protectedDiv(x, 1), sub(x, -1))))))))))), sub(cos(sin(sin(sin(add(mul(neg(protectedDiv(add(x, -1), add(1, 0))), 0), sub(protectedDiv(x, 1), -1))))), add(neg(add(sin(protectedDiv(x, 0)), mul(neg(0), neg(0)))), sub(cos(protectedDiv(cos(neg(mul(1, x))), cos(cos(cos(x))))), protectedDiv(mul(neg(cos(x)), neg(0)), sub(neg(x), cos(sub(1, neg(1))))))))), cos(cos(add(add(sin(add(protectedDiv(sin(protectedDiv(sub(x, x), protectedDiv(x, x))), add(mul(neg(x), neg(x)), protectedDiv(sub(x, -1), add(x, sub(1, x))))), neg(mul(sub(protectedDiv(1, x), cos(1)), mul(cos(x), cos(1))))), x), protectedDiv(mul(sin(neg(cos(cos(1)))), x), x)))))

2) Island Model Solution Tree Code:

sub(sin(mul(x, x)), sin(neg(mul(cos(add(neg(x), cos(neg(x)))), cos(mul(neg(sin(protectedDiv(protectedDiv(x, -1), -1))), mul(neg(cos(protectedDiv(neg(cos(protectedDiv(x, sin(mul(x, x))))), protectedDiv(x, sin(x))))), protectedDiv(neg(x), protectedDiv(sin(protectedDiv(0, -1)), 1)))))))