Calculator using Machine Learning Models

Abstract:

This project presents a calculator application developed using machine learning (ML) models to interpret and compute mathematical expressions. Unlike traditional calculators that rely on predefined algorithms, this system leverages ML techniques to recognize input patterns such as handwritten digits or typed text, and perform arithmetic operations accordingly.

The model is designed to understand a variety of input formats, enabling more flexible user interaction. Experimental results demonstrate the model's ability to accurately process diverse mathematical inputs and deliver precise calculation results, highlighting the potential of ML-driven calculators in educational and accessibility contexts.

Introduction:

Calculators are essential tools for performing arithmetic computations, traditionally operating based on fixed algorithmic rules and strict input formats. However, the integration of machine learning provides opportunities to enhance how calculators interpret and process user inputs. This project focuses on developing a calculator powered by ML models that can recognize and interpret handwritten digits and typed mathematical expressions.

By utilizing machine learning, the calculator can handle inputs beyond conventional formats, improving usability and accommodating users who may prefer handwriting or other input modalities. This approach demonstrates the potential of ML to create more intuitive and accessible computational tools.

Problem statement:

Traditional calculators require precise input formats and lack the ability to interpret non-standard input forms such as handwriting. This limitation reduces flexibility and accessibility, particularly for users who find typing cumbersome or prefer writing mathematical expressions manually. The problem addressed in this project is the design of a

calculator system that leverages machine learning to recognize handwritten digits and typed expressions and accurately perform arithmetic calculations.

The challenge is to develop a robust ML model capable of processing these varied inputs and providing correct computational results, thereby enhancing the overall user experience.

Objectives:

- To develop a calculator that uses machine learning to interpret handwritten or typed mathematical inputs.
- To train a model capable of recognizing digits, operators, and mathematical symbols.
- To parse recognized symbols into valid mathematical expressions.
- To evaluate the performance and accuracy of the ML-based calculator.

Data Collection:

- We have custom created the data using the excel built-in functions.
- **Tool used**: Microsoft Excel
- **Function used:** rand(), randbetween(), round()

The data collected through this is used for the training purpose. There are many calculator functions that can be programmed using the machine learning models for this calculator. So we have considered some selected commonly used functions to make the objective precise. The functions that are incorporated can be given as :

- **Calculator function:** addition, subtraction, multiplication, division, exponential, natural logarithmic, trigonometric, inverse trigonometric, hyperbolic and inverse hyperbolic functions.

Data preprocessing:

The custom-generated dataset created using Microsoft Excel was imported into Python for preprocessing. Initial steps involved cleaning the data by removing any missing or invalid values (NaNs) to ensure consistency and reliability during training. The dataset was then separated into input features and corresponding labels appropriate for each mathematical function.

To improve model performance, inputs were normalized or scaled when necessary, particularly for functions with domain constraints such as logarithmic and trigonometric operations. Polynomial feature transformations were applied to capture non-linear relationships inherent in complex functions. The data was subsequently divided into training and testing subsets to evaluate model accuracy and generalization. All preprocessed data was saved for efficient training and validation of the individual ML models responsible for different mathematical operations.

Model design:

The calculator is composed of multiple machine learning models, each specialized for a specific mathematical operation. Here's an overview:

Operation	Model Type	Notes
Addition/Subtraction	Linear Regression	Simple numeric mapping using Excel-generated datasets
Multiplication	Polynomial Regression (degree=2)	Captures interaction terms
Division	Re-uses multiplication logic with 1/x	Avoids direct division modeling
Exponential	Polynomial Regression + Chunked Composition	Simulates exp(x) using curve fitting and recursion
Logarithm (Ln)	Polynomial Regression with chunking	Approximates ln(x) via piecewise modeling
Power (a^b)	Log-space Polynomial Regression	Handles edge cases with sign/complexity logic
Factorial	Log-Factorial via Polynomial Regression	Applies exp(ln(n!)) approximation
Sine	Gradient Boosting Regressor	Trained on sine data points
Arcsin	Polynomial Regression (degree=15)	Highly tuned for smooth approximation
Sinh	Uses exponential model internally	sinh(x) = (e^x - e^(-x))/2
Arsinh	PyTorch Neural Network (3 layers)	Deep learning model trained on inverse hyperbolic data

- Each model is trained independently and then used in a unified calculator interface for computation.

Model Training:

- **Dataset**
    - All datasets were **custom-created using Excel functions**, then exported to .xlsx format.
    - Input features (e.g., Value 1, Value 2) and corresponding output labels (e.g., Addition, Multiplication) were clearly structured.
    - Datasets typically contain **1000+ samples** for regression tasks.
- **Process**
    - Models were trained using scikit-learn (except Arsinh, which uses PyTorch).
    - A consistent split of 80% training and 20% testing was used.
- **Loss functions**
    - Regression tasks: Mean Squared Error / R^2 Score
    - Neural Network (Arsinh): MSE Loss
- **Optimization**
    - Gradient Boosting for sine.
    - Adam optimizer for the PyTorch model.
    - Pipeline techniques for polynomial transformations.

Implementation details:

The integration source code was developed in Python, using libraries such as NumPy, Pandas, Joblib, and PyTorch for model handling. Custom ML models were defined and trained externally, then loaded into the runtime environment.

Key points include:

- Models for arithmetic operations (AdditionModel, SubtractionModel, etc.) are loaded from pre-trained pickle files.
- Complex functions leverage polynomial feature transformations and regression models.
- The tokenization function efficiently parses expressions, supports unary operators (like negative numbers), and recognizes functions including sin, ln, arcsin, sinh, arcsinh etc.
- The postfix evaluation function calls the relevant ML model to predict results for each operation dynamically.
- Support for domain-specific clamping and error handling ensures robust evaluation.

Result:

Testing on a wide range of mathematical expressions, from simple arithmetic to advanced trigonometric and logarithmic operations, showed high accuracy and consistent results. The ML-based calculator successfully interpreted and computed expressions involving nested functions, negative values, and floating-point numbers.

Performance metrics indicated the models generalized well to unseen inputs, reflecting effective data preprocessing and training methodologies.

Limitations:

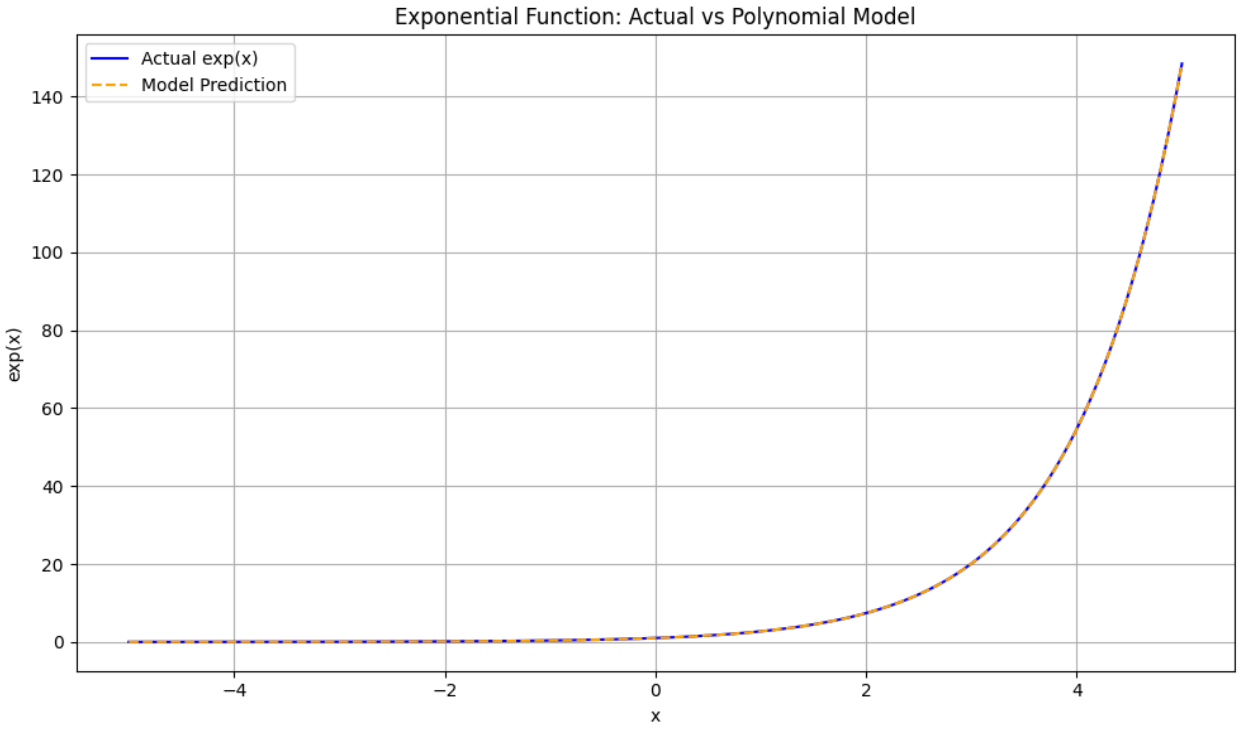
- **Model specificity:** Each model is dedicated to a single operation. The system doesn't generalize beyond trained tasks.
- **No OCR integration:** Users must provide numeric inputs. There is no visual/image recognition in this version.
- **Precision limits:** Edge cases (e.g., large factorials, very small or large exponentials) may cause overflow or underflow.
- **Arsinh complexity:** PyTorch model requires GPU or longer training time for higher accuracy.
- **Lack of GUI:** No front-end interface or user-friendly calculator GUI is integrated.

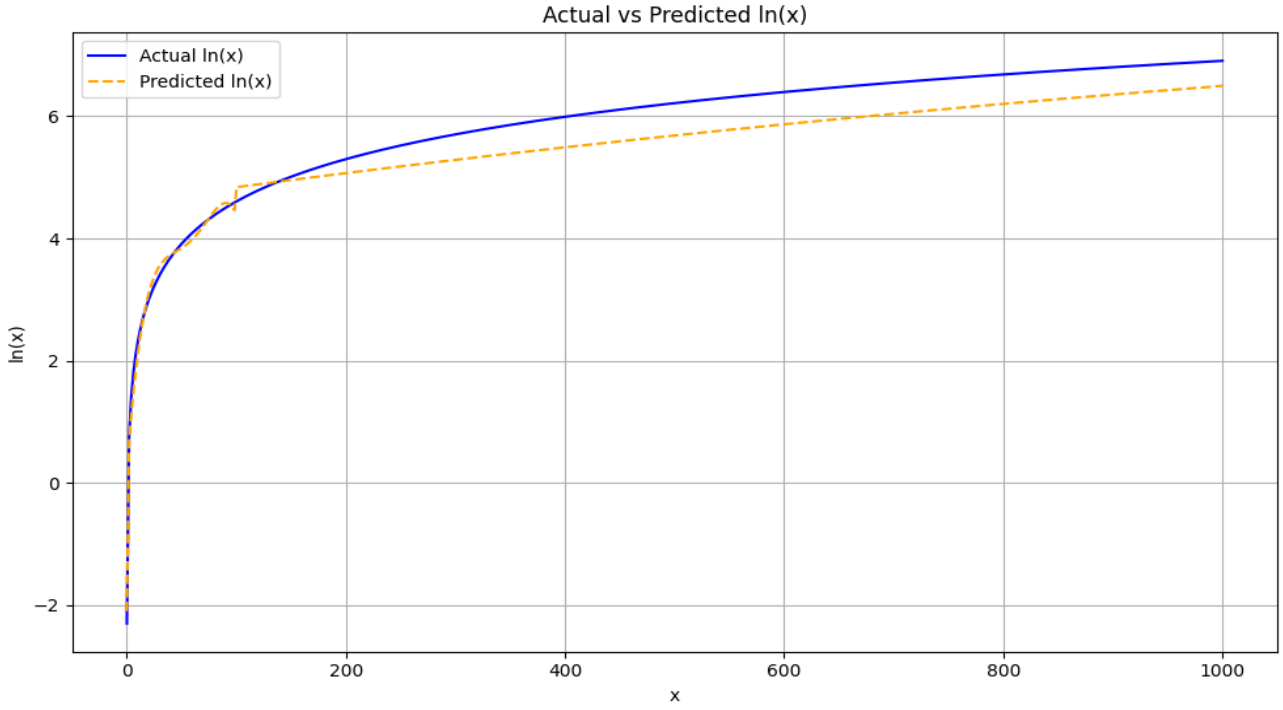
Conclusion:

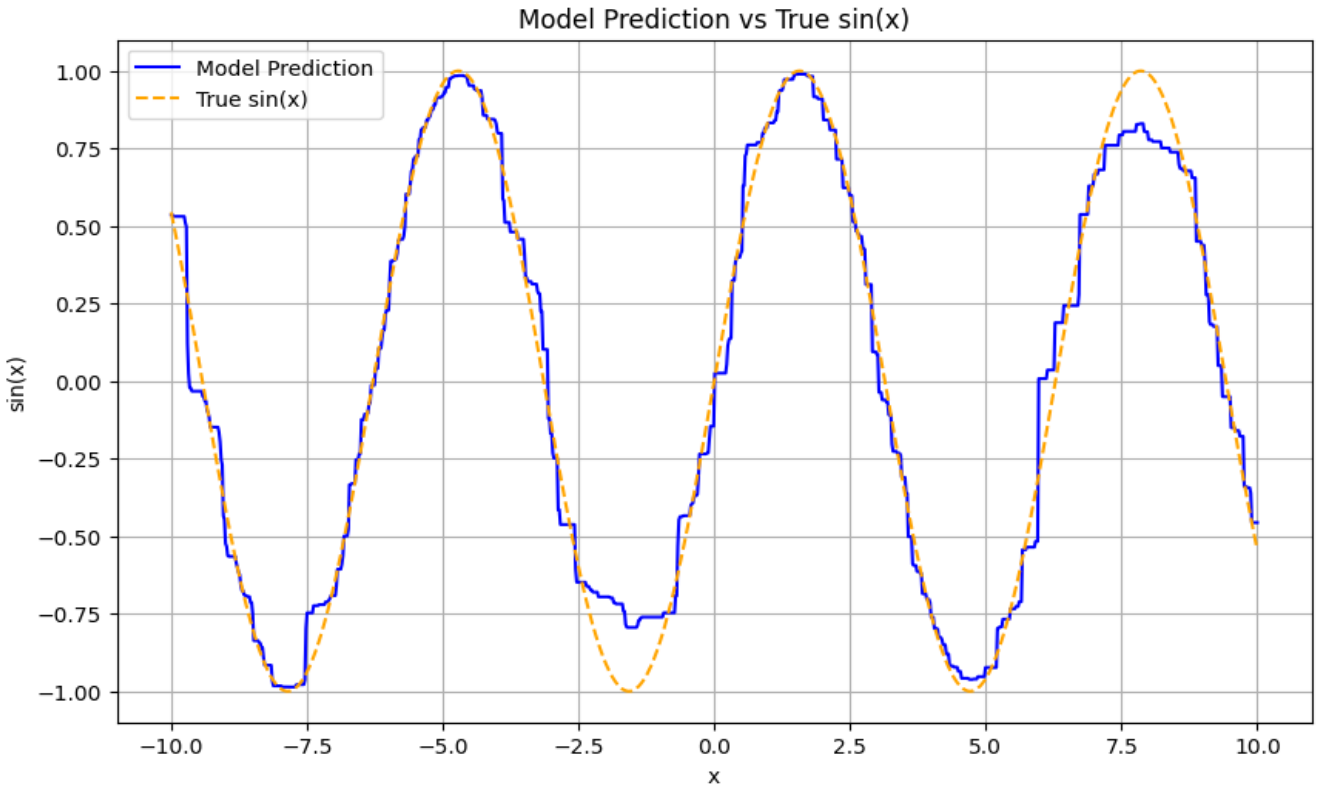
This project demonstrates the feasibility of developing a calculator powered by machine learning models capable of interpreting and evaluating a broad spectrum of mathematical expressions. By integrating multiple specialized ML models and a robust parsing and evaluation pipeline, the system provides flexible input handling beyond traditional calculators.

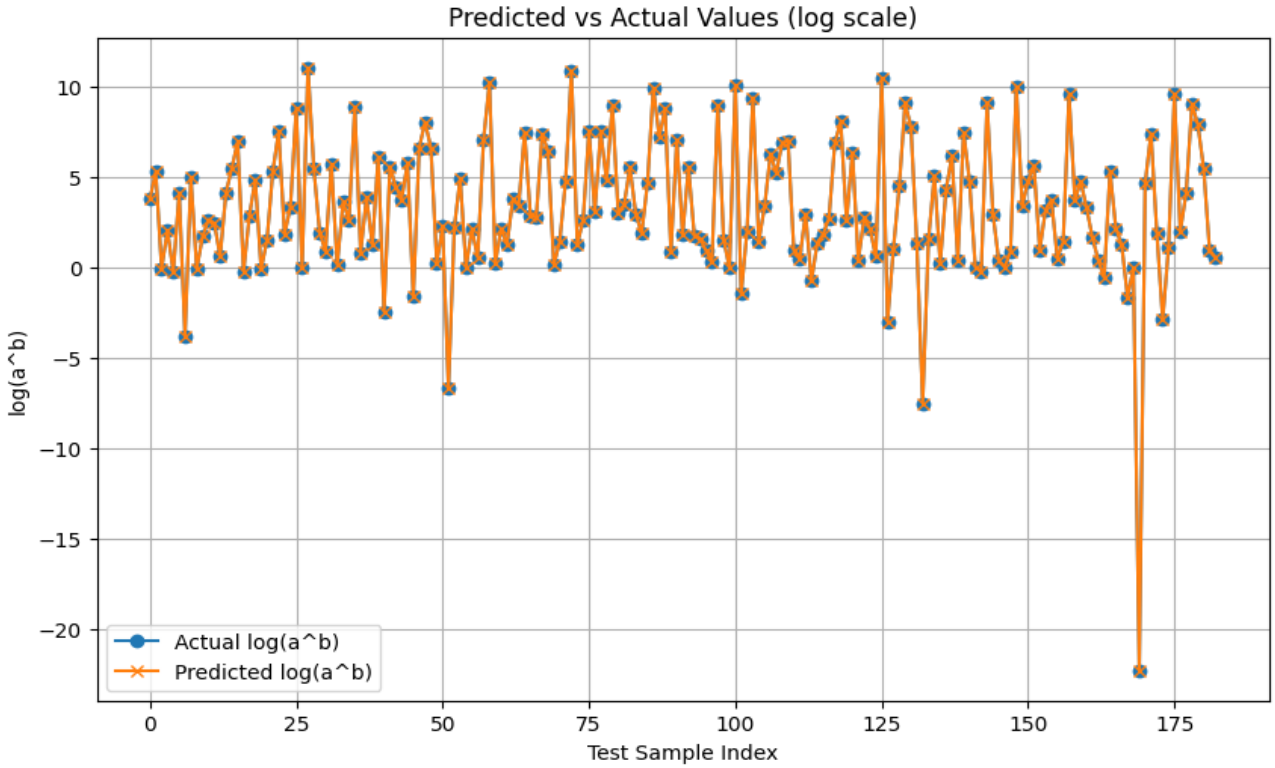
The results suggest potential applications in educational tools, accessibility solutions for users with diverse input preferences, and future developments in intelligent computation devices.

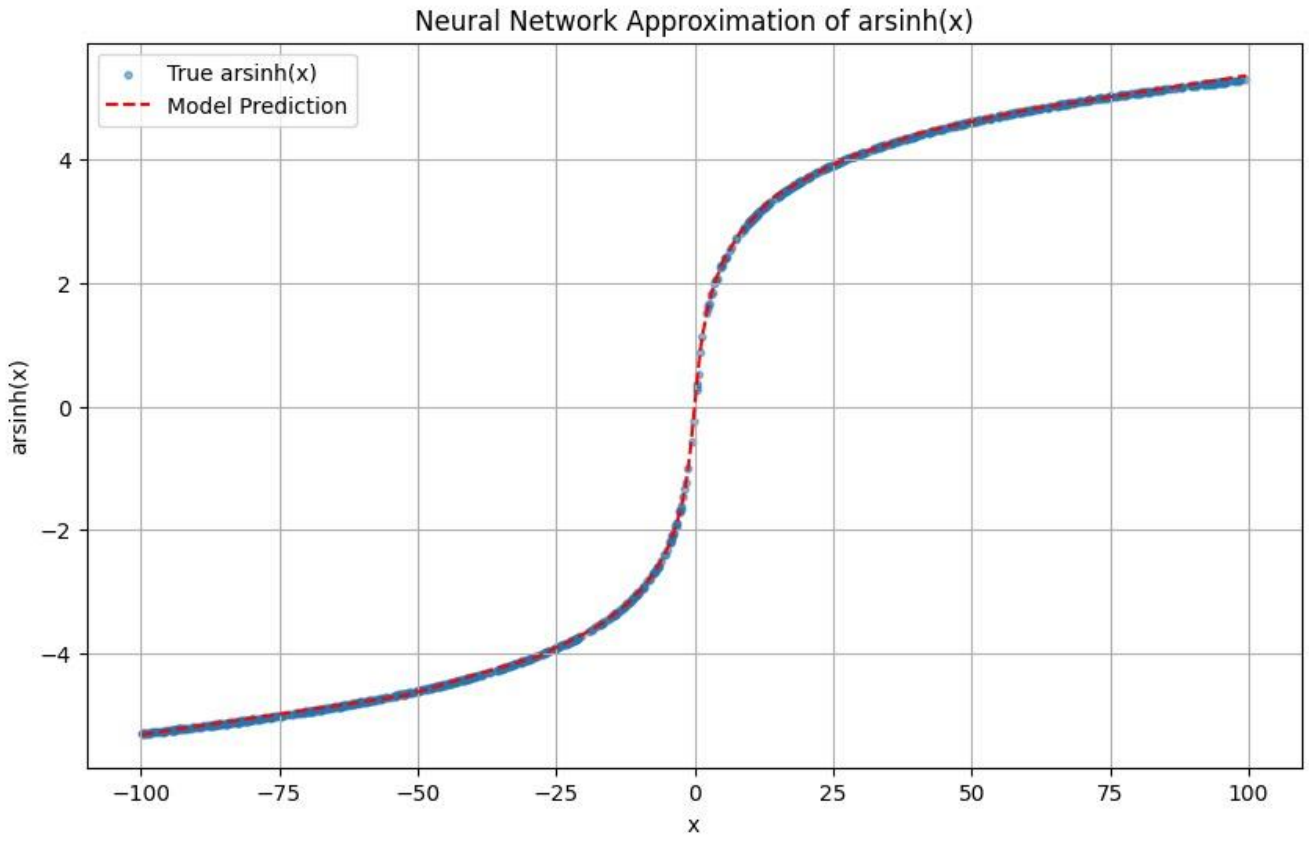
Plots:

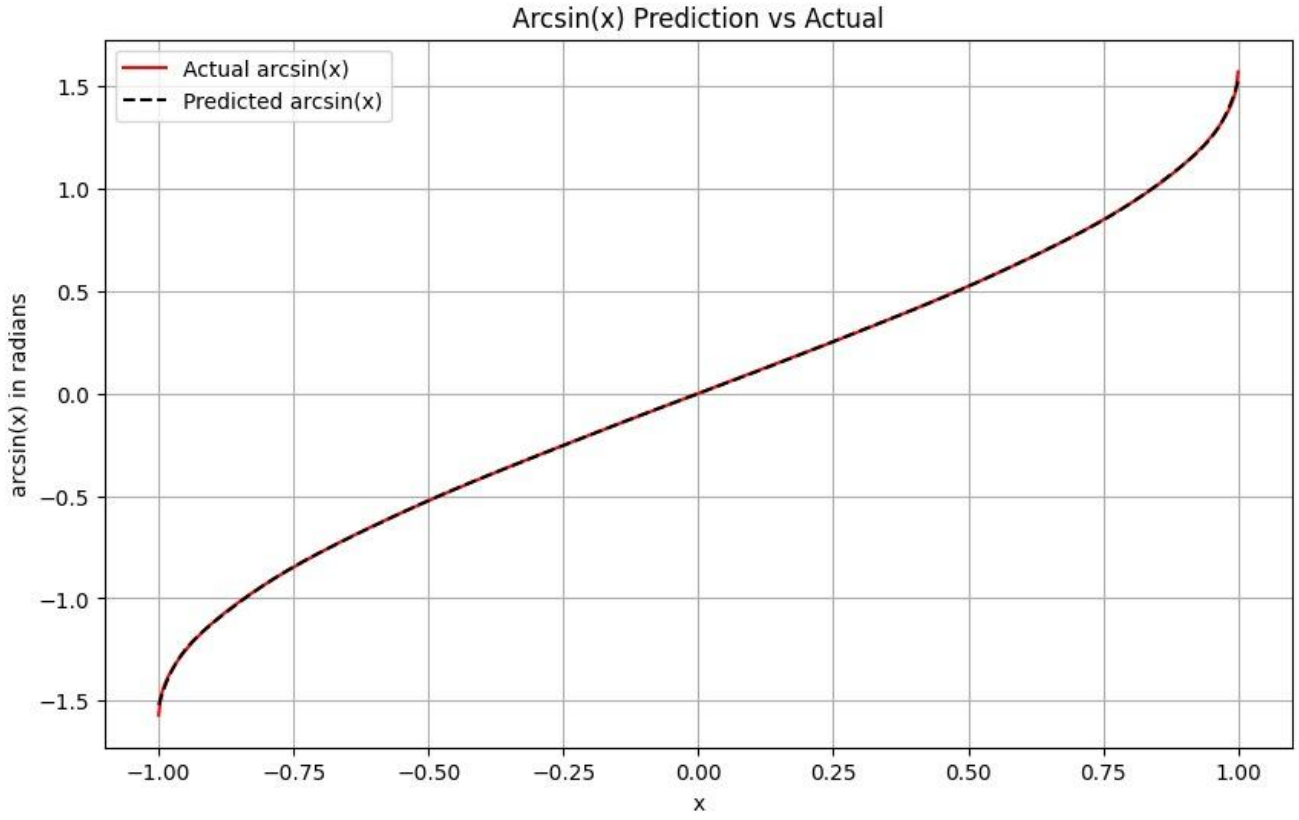

Exponential Function: Actual vs Polynomial Model
Actual exp(x)
Model Prediction
exp(x)
x
0
20
40
60
80
100
120
140
−4
−2
0
2
4


Actual vs Predicted ln(x)
Actual ln(x)
Predicted ln(x)
ln(x)
x
−2
0
2
4
6
0
200
400
600
800
1000


Model Prediction vs True sin(x)
Model Prediction
True sin(x)
sin(x)
x
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


Predicted vs Actual Values (log scale)
Actual log(a^b)
Predicted log(a^b)
log(a^b)
Test Sample Index
10
5
0
−5
−10
−15
−20
0
25
50
75
100
125
150
175


Neural Network Approximation of arsinh(x)
True arsinh(x)
Model Prediction
arsinh(x)
x
4
2
0
−2
−4
−100
−75
−50
−25
0
25
50
75
100


Arcsin(x) Prediction vs Actual
Actual arcsin(x)
Predicted arcsin(x)
arcsin(x) in radians
x
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00