Kathmandu University

Department of Computer Science and Engineering

Dhulikhel , Kavre

A Mini-Project Report

on

"**Linear Regression Visualizer"**

[Subject Code: COMP342]

Submitted by:

Parth Pandit (45)

Sammit Poudyal (50)

Submitted to:

Mr. Dhiraj Shrestha

Department of Computer Science and Engineering

Submission Date:

17/01/2026

Acknowledgement

We would like to express our sincerest gratitude to our computer graphics instructor Mr. Dhiraj Shrestha for this wonderful opportunity to create an application that helps us explore the concepts that are mentioned in our course. We would like to give him our special thanks for his guidance, supervision, and encouragement throughout our project's duration.

Abstract

This project presents an interactive visualization tool for learning linear regression training and gradient-based optimization, developed in Python using Pygame and PyOpenGL. The system generates a synthetic noisy dataset from an underlying "true" line and visualizes, in real time, the relationship between training points, the current predicted regression line, and the ground-truth reference. Through an intuitive OpenGL-rendered interface, users can control the learning process with start/pause, single-step iteration, reset, and data regeneration, while also adjusting the animation delay to observe parameter updates at different speeds. The visualizer supports multiple optimizers-SGD, Momentum, and Adam-allowing direct comparison of convergence behavior under identical conditions. To enhance conceptual understanding, optional visual aids display residuals, an on-screen MSE history plot, and an educational panel showing the core update equations and legend. Overall, the application serves as an effective educational environment for understanding how gradient descent optimizes slope and intercept parameters, and how optimizer choice impacts convergence in practical machine learning workflows.

Keywords: *Linear Regression visualization, Gradient Descent Visualization, Optimization Algorithms*

Table of Contents

List of Figures

Acronyms/Abbreviations

CPU:	Central Processing Unit
GPU:	Graphical Processing Unit
GUI:	Graphical User Interface
IDE:	Integrated Development Environment
OpenGL:	Open Graphics Library
OS:	Operating System
PyGame:	Python Game Development Library
PyOpenGL:	Python Open Graphics Library
UI:	User Interface
SGD:	Stochastic Gradient Descent
MGD:	Momentum Gradient Descent

Introduction

1.1 Introduction

Linear Regression Visualizer is an interactive educational application designed to facilitate a deeper understanding of the training phases in linear regression through real-time visual representation. Developed in Python utilizing the OpenGL library for high-performance graphics rendering and Pygame for comprehensive window and event management, the tool allows users to observe a model iteratively fitting a regression line to stochastic datasets via gradient-based optimization. The interface is engineered to bridge the gap between abstract mathematical formulas and practical application by highlighting key concepts through dynamic visual elements, including plotted data points, shifting predicted lines, and a live Mean Squared Error (MSE) convergence chart.

Created as a core component of a Computer Graphics course, this project integrates machine learning foundations, numerical computing, and human-computer interaction into a unified learning platform. The system addresses common pedagogical challenges by transforming the static training procedures of gradient descent into an intuitive, interactive experience. Users are afforded full control over the visualization process through features such as play/pause, step-by-step navigation, and adjustable execution speeds, alongside the ability to compare different optimizers like SGD, Momentum, and Adam. While the current implementation focuses on univariate regression, its modular architecture ensures extensibility for future integration of more complex models and datasets.

1.2 Background

Linear regression serves as a cornerstone of statistical modeling and machine learning, providing a vital framework for variable relationship analysis and predictive modeling. Despite its foundational importance, the conceptual mechanics of model training—such as gradient-based optimization, loss function minimization, and parameter convergence—are often obscured by static pedagogical methods. Traditional learning resources typically rely on fixed graphs and complex mathematical derivations, which can fail to convey the dynamic "in-between" stages of the learning process. Consequently, abstract concepts like learning rate

influence and the gradual refinement of weights can remain difficult for students to fully conceptualize without a means to observe these transitions in real time.

To address these educational challenges, this project introduces an interactive visualization platform that transforms the linear regression training procedure into a controllable, high-fidelity visual experience. By utilizing OpenGL to render real-time updates of the evolving regression line, residual errors, and the Mean Squared Error (MSE) curve, the system allows users to directly observe the correlation between mathematical updates and their visual outcomes. Furthermore, the inclusion of multiple optimizers—specifically SGD, Momentum, and Adam—enables a comparative analysis of how different optimization strategies affect convergence stability and speed. Ultimately, this tool aims to bridge the gap between theoretical machine learning foundations and practical implementation, providing a hands-on environment that reinforces core concepts in both numerical computing and computer graphics.

1.3 Objectives

The objectives of our project are as follows:

1. To develop an interactive environment for visualizing linear regression training through real-time gradient-based optimization.
2. To implement a synthetic data generation system with configurable noise levels to observe model fitting across various distributions.
3. To integrate execution controls and optimizer switching (SGD, Momentum, and Adam) for a comparative study of convergence behavior and speed.

System Analysis

2.1 General Block Diagram

The general block diagram of our application is depicted in Figure 2-1.

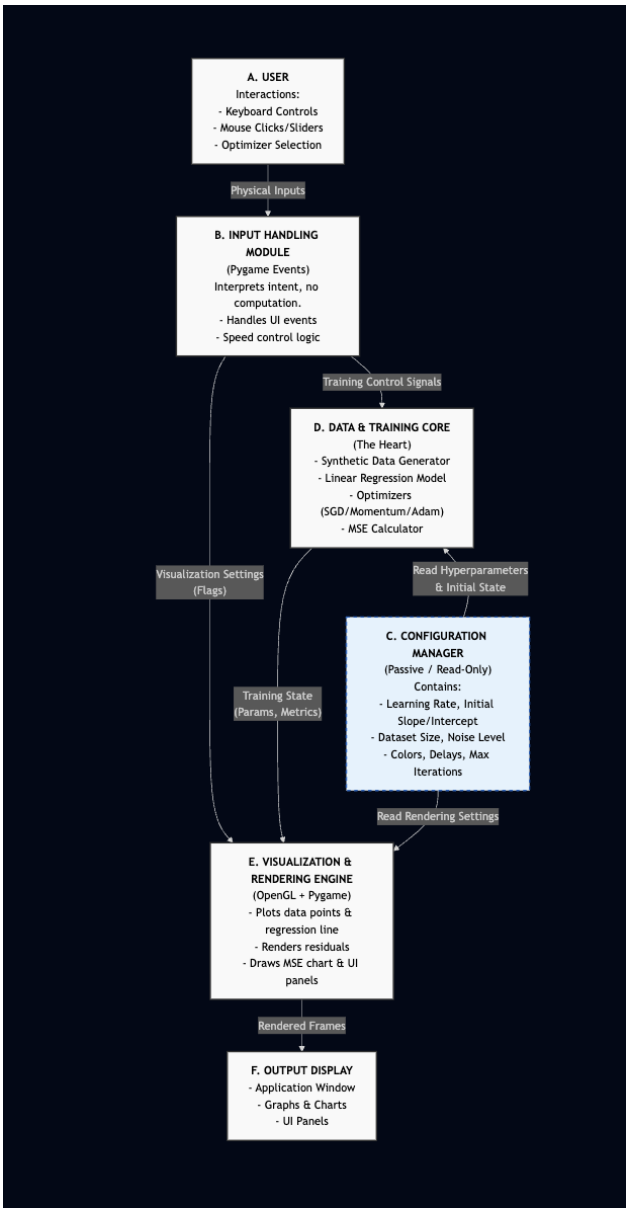

Figure 2--1: General Block Diagram

2.2 System Architecture Diagram

The system architecture diagram of our application is depicted in Figure 2-2.

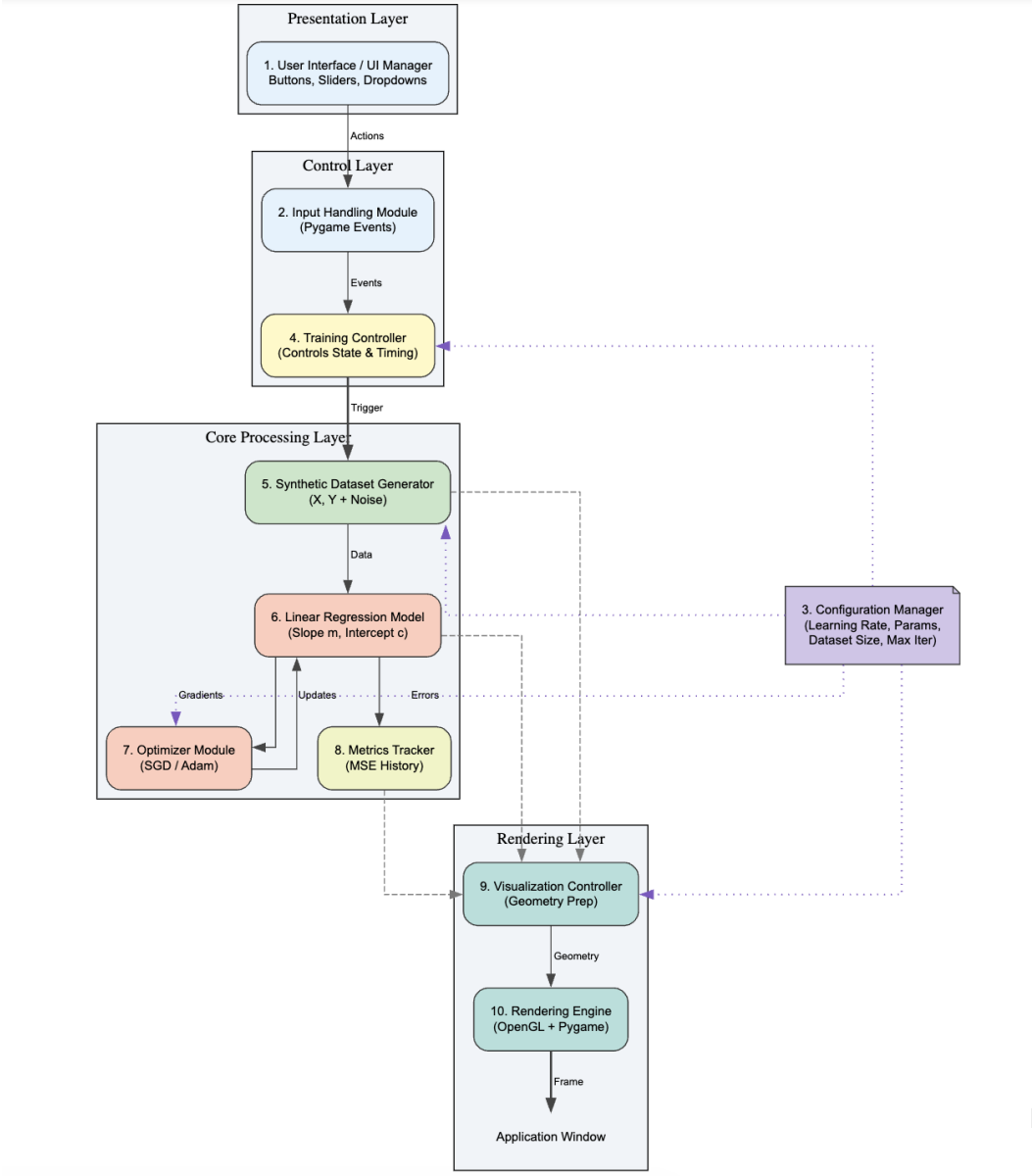

Figure 2-2: System Architecture Diagram

2.3 Training Flow Diagram

The training flow diagram of our application is depicted in Figure 2-3.

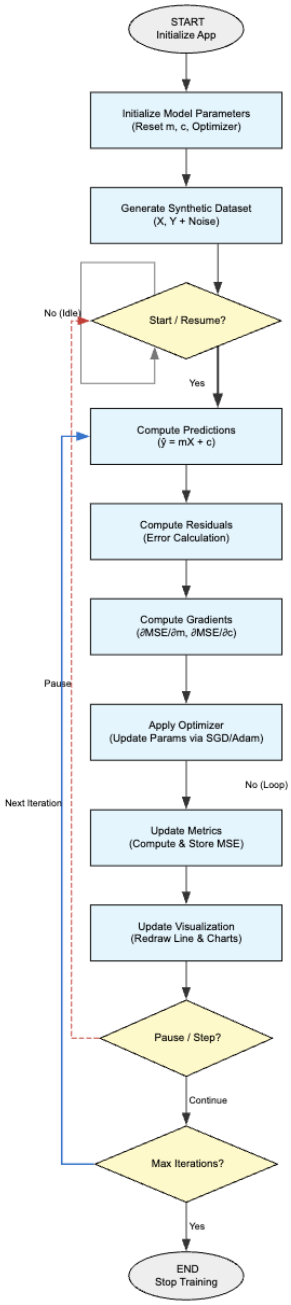

Figure 2-3: Training Flow Diagram

2.4 Software Requirements

The software requirements of our application are as follows:

- Python Environment: Python 3.10 or higher (functionality verified on version 3.12).
- Required Libraries:
  - NumPy: For efficient numerical computations and data handling.
  - scikit-learn: For dataset generation and utility functions.
  - Pygame: For window management and handling user input.
  - PyOpenGL & PyOpenGL_accelerate: For high-performance 2D/3D graphics rendering.

2.5 Hardware Requirements

The recommended hardware requirements for our application are provided below:

- Processor: A modern Intel or AMD CPU.
- Memory: Minimum 4 GB RAM (8 GB recommended for optimal performance).
- Graphics: An integrated or dedicated GPU with support for OpenGL.
- Display: A minimum screen resolution of 1366×768 for proper UI layout (1920×1080 preferred).
- Storage: Approximately 200 MB or more free disk space for Python, project dependencies, and source files.
- Drivers: Updated GPU drivers to ensure stable OpenGL context creation and consistent rendering performance.

Mathematical Model and Formulation

3.1 Overview

This chapter presents the mathematical foundation of the Linear Regression Visualizer. The system models a univariate linear regression, where the goal is to learn parameters m (slope) and c (intercept) that best fit a dataset. Training is performed using gradient-based optimization and visualized in real time.

3.2 Linear Regression Model

The regression hypothesis is:

$$\hat{y}=mx+c$$

where x is the input, y is predicted output, m is the slope, and c is the intercept. The objective is to minimize the prediction error over all data points.

3.3 Loss Function: Mean Squared Error (MSE)

The system uses MSE to quantify prediction error:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

- n = number of data points
- y_i= true output
- $\hat{y}_i$ = predicted output

MSE is differentiable and convex, suitable for gradient-based optimization.

3.4 Gradient Computation

Gradients of the MSE w.r.t parameters:

$$\frac{\partial MSE}{\partial c} = \frac{2}{n}\left(\sum_{i=1}^{n}(\hat{y}_i - y_i)\right)$$

$$\frac{\partial MSE}{\partial m} = \frac{2}{n}\left(\sum_{i=1}^{n}(\hat{y}_i - y_i)x_i\right)$$

These indicate the direction and magnitude to adjust *mm*m and *cc*c.

3.5 Gradient Descent Update Rule

Parameter updates are performed iteratively:

$$\theta_{t+1} = \theta_t - \alpha \nabla L(\theta_t)$$

- θ = m or c
- α = learning rate
- ∇L = gradient of MSE

3.6 Optimization Algorithms

3.6.1 Stochastic Gradient Descent (SGD)

$$\theta = \theta - \alpha g$$

- Simple direct update
- Can oscillate, slower convergence

3.6.2 Momentum Optimizer

$$v_t = \beta v_{t-1} - \alpha g_t \, , \, \theta_t = \theta_{t-1} + v_t$$

- Introduces velocity term
- Reduces oscillation, accelerates convergence

3.6.3 Adam Optimizer

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t$$
$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(g_t)^2$$
$$\theta_t = \theta_{t-1} - \frac{\alpha \, m_t}{\sqrt{v_t} + \epsilon}$$

- Combines momentum and adaptive learning rates
- Stable, fast convergence

3.7 Synthetic Dataset Generation

Data is synthetically generated as:

$$y = m_{true} x + c_{true} + \epsilon$$

- m_{true} , c_{true} underlying slope and intercept
- ϵ = Gaussian noise

This allows controlled observation of parameter convergence and effect of noise.

3.8 Convergence Behavior

As training progresses:

- Gradients approach zero
- MSE decreases
- Regression line stabilizes toward the true underlying relationship

Different optimizers exhibit distinct speed and smoothness of convergence, visualized in real time.

System Implementation

4.1 Configurations

Our system utilizes a centralized configuration structure to manage all adjustable parameters and visual settings, ensuring high modularity and ease of customization. The primary configuration categories are as follows:

1. Canvas and Viewport Configuration
    - Primary Canvas Size: 1100 x 700 pixels
    - MSE Chart Size: 1100 x 180 pixels
    - Padding: 80 pixels Note: The system supports dynamic coordinate mapping to maintain visual consistency across the interface.
2. Model and Training Parameters
    - Default Learning Rate: 0.01
    - Dataset Size: 50 points
    - Synthetic Noise (Standard Deviation): 1.5
    - Initial Model State: Slope (0.0), Intercept (5.0)
    - Reference Target: Slope (1.5), Intercept (2.0)
3. Visualization and Aesthetic Parameters
    - Color Palette: Deep slate gradients for background, cyan for axes, and high-contrast orange for the regression line.
    - Point Styling: Dual-tone blue (fill and outline) for data points.
    - UI Elements: Distinct visual indicators for residual (error) lines and grid alignment.
    - Typography: Standardized use of Helvetica for primary text, bold labels, and titles.
4. Execution and Animation Configuration
    - Default Training Delay: 20 ms
    - Maximum Iteration Limit: 500 steps
    - Control Logic: Real-time adjustment of training speed and playback state.

```
# Visualization and model configuration
CANVAS_W, CANVAS_H, PAD = 1100, 700, 80
MSE_CANVAS_W, MSE_CANVAS_H = 1100, 180
LEARNING_RATE = 0.01
TRUE_S, TRUE_I = 1.5, 2.0
INITIAL_S, INITIAL_I = 0.0, 5.0
DELAY = 20
MAX_ITER = 500
N_POINTS = 50
NOISE_STD = 1.5

BG_TOP, BG_BOTTOM = "#0f172a", "#1e293b"
GRID_COLOR, AXIS_COLOR = "#243453", "#5eead4"
POINT_FILL, POINT_OUTLINE = "#60a5fa", "#0ea5e9"
LINE_COLOR, LINE_GLOW = "#f97316", "#fb923c"
RESIDUAL_COLOR = "#cbd5e1"
TEXT_COLOR = "#e2e8f0"
FONT_MAIN = ("Helvetica", 12)
FONT_BOLD = ("Helvetica", 12, "bold")
FONT_TITLE = ("Helvetica", 14, "bold")
```

Figure 3-1: Configuration File

4.2 Algorithms

4.2.1 Linear Regression Training

The system utilizes a gradient-based iterative approach to minimize the Mean Squared Error (MSE). The training procedure follows these sequential steps:

1. Initialization: Reset the training state, clear historical performance metrics, and initialize parameters (slope and intercept) based on the current configuration.
2. Dataset Loading: Generate or reload the synthetic dataset consisting of noisy data points (X, Y).
3. Forward Pass: Calculate predictions for the entire batch: y_pred = slope * X + intercept.
4. Error Calculation: Determine the prediction residuals: error = y_pred - Y.

5. Gradient Computation: Calculate the partial derivatives for both parameters:
    - d_intercept = (2/n) * sum(error)
    - d_slope = (2/n) * sum(error * X)
6. Parameter Update: Apply the selected optimization rule (SGD, Momentum, or Adam) to update the slope and intercept.
7. Metrics Recording: Compute the current MSE and append it to the history log for real-time plotting.
8. Iteration: Repeat steps 3–7 until the maximum iteration threshold is reached or a user interrupt occurs.

4.2.2 Optimization Algorithm

The project implements various parameter update rules to demonstrate differing convergence behaviors:

1. Stochastic Gradient Descent (SGD) A standard update rule where parameters are adjusted directly by the gradient scaled by the learning rate (alpha):
    - theta = theta - alpha * d_theta
2. Momentum Optimizer Introduces a velocity term (v) and a friction coefficient (beta) to accelerate gradients in the right direction and dampen oscillations:
    - v_theta = beta * v_theta - alpha * d_theta
    - theta = theta + v_theta
3. Adam (Adaptive Moment Estimation) Utilizes first (m) and second (v) moment estimates to adapt the learning rate for each parameter individually. This involves bias-correction and a small constant (epsilon) to prevent division by zero:
    - m_t = beta1 * m_t-1 + (1 - beta1) * g_t
    - v_t = beta2 * v_t-1 + (1 - beta2) * g_t^2
    - theta_t = theta_t-1 - (alpha * m_hat) / (sqrt(v_hat) + epsilon)

4.2.3 Synthetic Dataset Generation

To provide a consistent yet adjustable environment for experimentation, the system generates data using the following logic:

- Input Generation: Create N evenly spaced inputs (X) across the visible canvas range.
- Target Calculation: Apply the true underlying linear relationship while adding Gaussian noise: Y = True_S * X + True_I + Noise.
- Viewport Mapping: Calculate data bounds (min/max) to ensure the points generated are correctly normalized and scaled for the OpenGL viewport.

Regenerating the dataset allows users to observe how varying levels of noise influence the final fitted line and the stability of the convergence curve.

4.3 Display Mechanism

4.3.1 Data Point Rendering

- The dataset points are rendered with OpenGL using the GL_POINTS primitive.
- To make the points appear smooth and circular, blending (GL_BLEND) and point smoothing (GL_POINT_SMOOTH) are enabled.
- A two-pass approach is used for clarity: first a larger point is drawn as an outline, then a smaller point is drawn on top as the fill, producing a bordered marker.

4.3.2 Line Rendering

- Lines are drawn using OpenGL's GL_LINES primitive after setting the required color and thickness with glColor4f and glLineWidth.
- The regression line is highlighted using a glow effect: a thick semi-transparent line is drawn first, followed by a thinner solid line.
- Residuals are visualized by drawing line segments between each real data point and its predicted point on the regression line, representing the error magnitude.
- Grid lines and axes are drawn with transparency to guide interpretation without dominating the main visualization.

4.3.3 Text and UI Rendering

- Text and UI elements are first rendered onto a Pygame surface using its font engine (since OpenGL does not directly render high-quality text easily).

- The surface is converted into RGBA pixel data and uploaded to the GPU as a texture using glTexSubImage2D.
- The UI is displayed by drawing a screen-space quad (two triangles) and mapping the texture onto it, resulting in crisp labels, buttons, and panels layered over the OpenGL scene.

4.4 Input Handling

The input handling component manages all keyboard and mouse interactions through Pygame's event system. It allows users to control training execution, adjust visualization settings, and interact with the on-screen UI.

4.4.1 Mouse Interactions

Mouse input is mainly used to interact with the UI panels and controls:

- Left click on UI buttons: Start training, pause training, step once, reset model, and regenerate data.
- Click on optimizer dropdown: Open/close the optimizer menu and select between SGD, Momentum, and Adam.
- Click-and-drag on delay slider: Increase or decrease the training delay (controls the animation/training speed.)

  Note: The main plot area is primarily for visualization; model interaction is performed through the UI controls.

4.4.2 Keyboard Controls

The keyboard controls are as follows:

SPACE: Start/Pause training

T: Start training

P: Pause training

S: Step once (single iteration update)

R: Reset model and metrics to initial state

N: Regenerate dataset and restart training state

1 / 2 / 3: Switch optimizer (SGD / Momentum / Adam)

V: Toggle residual (error) visualization

F: Toggle formulas/legend panel

[/]: Decrease / increase training speed by adjusting delay

ESC: Exit application

4.4.3 Button Panel

The bottom button panel provides the following actions:

- Start Training, Pause Training
- Step Once
- Reset Model
- Regenerate Data

The left panel provides a delay slider and a residual toggle, while the right panel provides an optimizer selection menu and educational reference content.

4.5 Visualization

The Linear Regression Visualizer is the main application module that coordinates all core components (data generation, training logic, rendering, and UI controls)

1. **Initialization:**
    a. Creates a Pygame window and initializes an OpenGL rendering context (double-buffered and resizable).
    b. Initializes the UI layout/manager, the dataset model, metric tracker, and the selected optimizer (SGD by default).
    c. Loads the initial training state by setting the initial slope/intercept and computing the starting metrics (so the UI shows values before the first step).
2. **Update Loop:**
    a. Continuously processes Pygame events (keyboard input, mouse clicks, window resize).
    b. Maintains a consistent frame rate using a clock and computes delta-time for time-based updates.
    c. Controls the training state (running/paused) and advances iterations either step-by-step or automatically based on the selected delay (speed).
3. **Rendering Pipeline:**
    a. Clears the frame and draws a gradient background to provide visual contrast.
    b. Renders the main plot: grid/axes, training data points, optional residuals,

the true reference line, and the current regression line.

c. Renders the MSE chart panel showing error history across iterations.

d. Draws the UI overlays (left controls, right reference panel, and bottom buttons) with interactive hover/click behavior, then swaps display buffers.

4. **State and Settings Management:**
    a. Handles dynamic settings such as delay (speed), residual visibility, and formula/legend visibility.
    b. Supports switching optimizers (SGD, Momentum, Adam) and resets optimizer internal state when required.
    c. Supports resetting the model (restart training from initial parameters) and regenerating the dataset (new noisy samples).
5. **Training Algorithm Integration:**
    a. Executes the linear regression update step by computing predictions, gradients, and optimizer-based parameter updates.
    b. Updates and records metrics (especially MSE) each iteration for plotting and status display.
    c. Ensures the visualization remains synchronized with the training process by updating the rendered line, residuals, and metric charts immediately after each step.

Results

This section contains all the interactions and results of our application:

Figure 5-1: Main Window

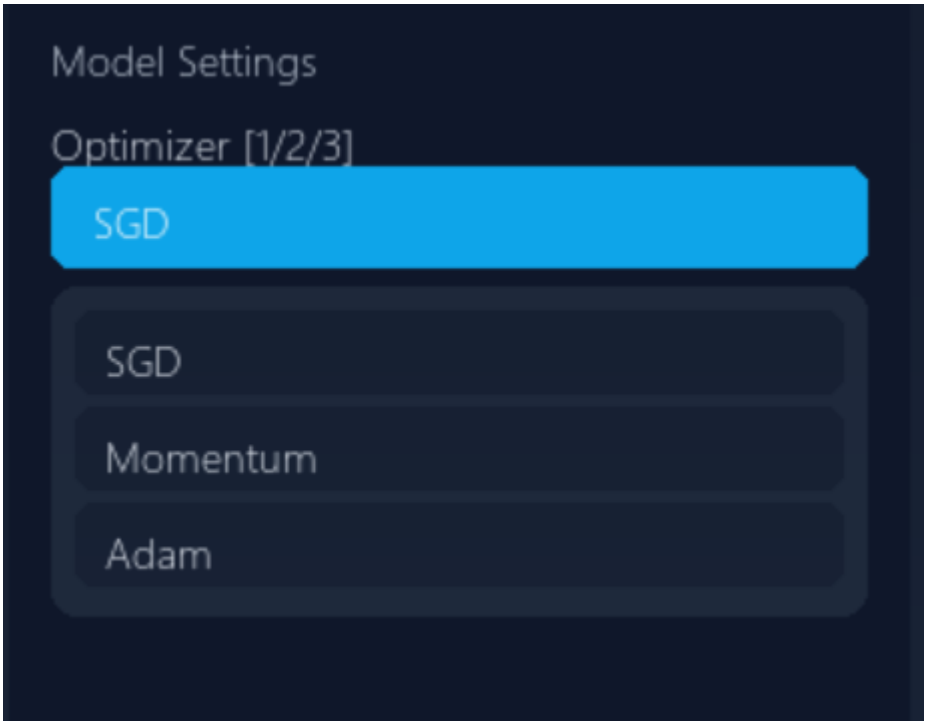

Figure 5-2: Optimizer Selection

Figure 5-3: Training Phase

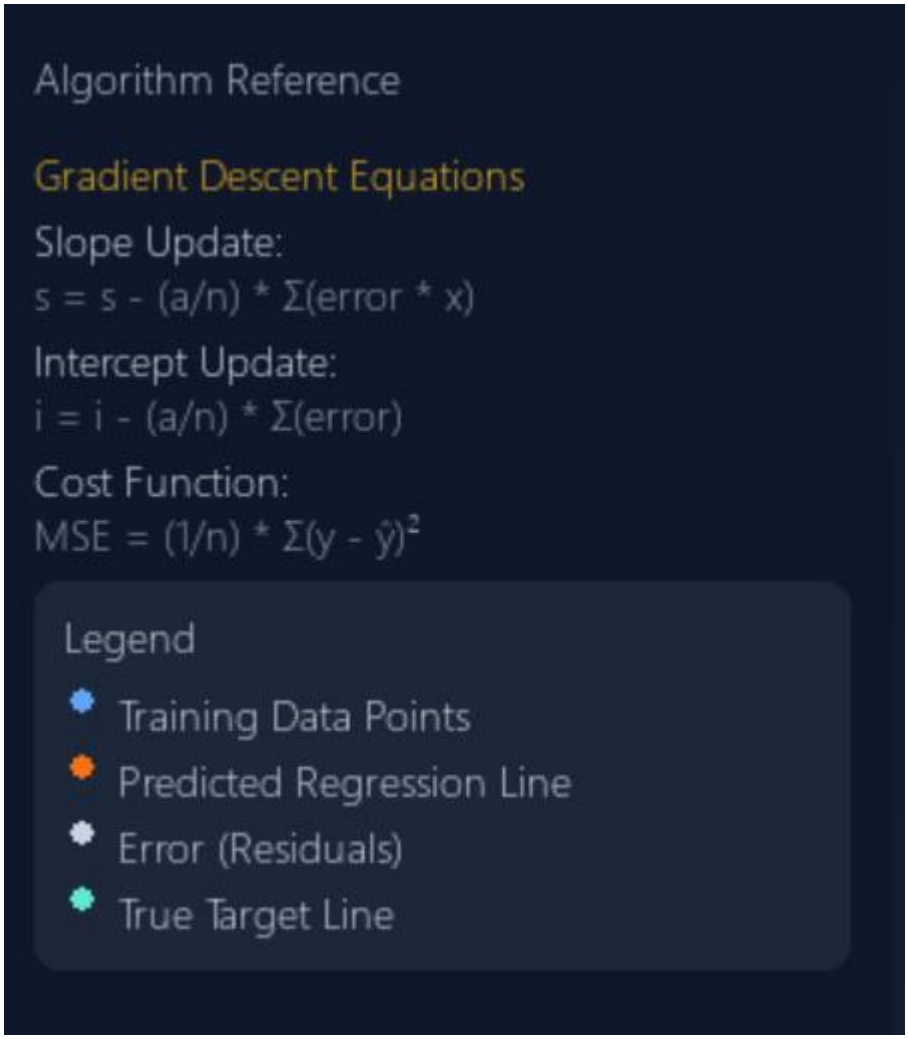

Figure 5-4: Algorithm Reference

Figure 5-5: Metrics During Runtime

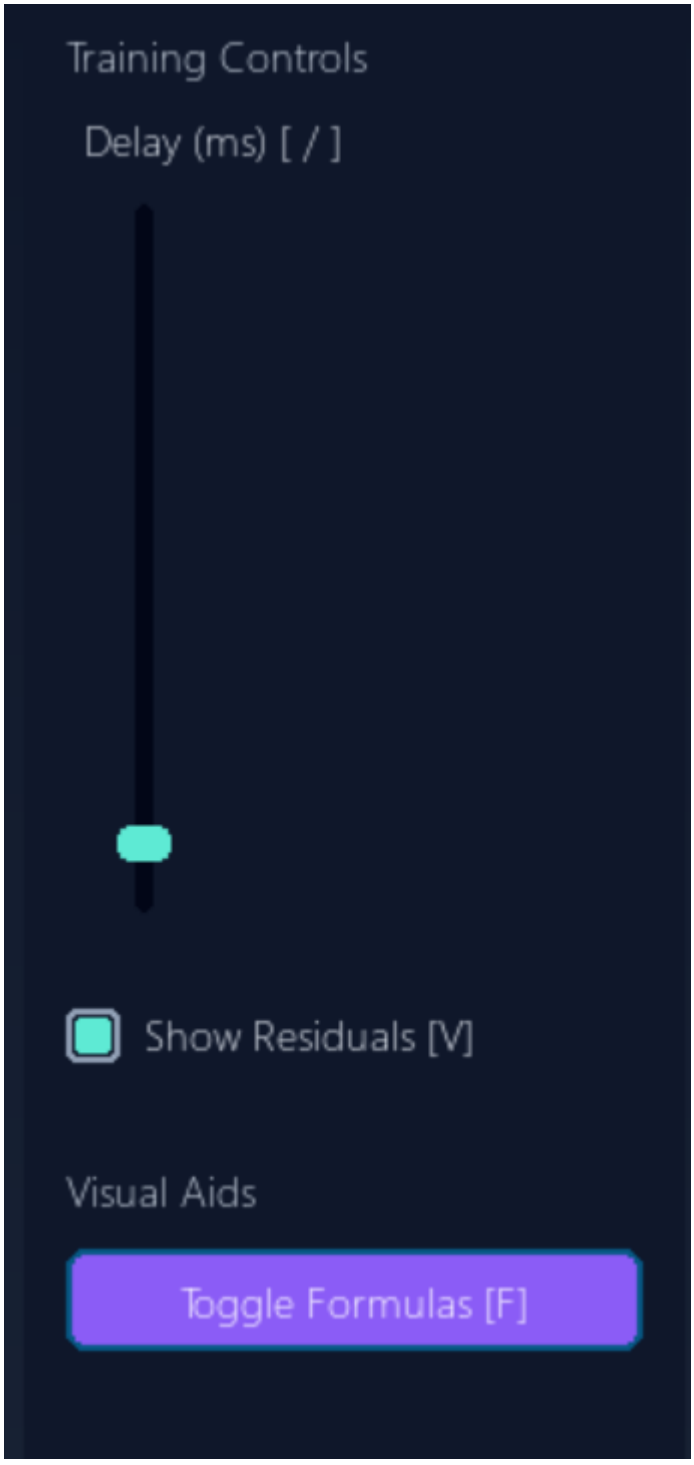

Figure 5-6: Training Controls

Figure 5-7: Execution Control Interface

Conclusion and Recommendation

Linear Regression (OpenGL) Visualizer successfully demonstrates how a linear model learns from data through gradient-based optimization in a clear and interactive manner. By combining real-time OpenGL rendering with Pygame-based input handling and UI overlays, the application makes abstract concepts such as parameter updates (slope/intercept), residual error, and loss minimization-visible as the training progresses. The inclusion of multiple optimizers (SGD, Momentum, and Adam), a live MSE chart, and step/pause/speed controls supports self-paced learning and help users compare convergence behavior under the same dataset conditions. Overall, the project meets its educational goal of improving understanding of regression training and optimization through an engaging visual and interactive experience.

6.1 Limitations

We experienced the following limitations while developing our application:

1. The application requires working OpenGL support, so it may not run on systems with missing/old GPU drivers.
2. It only visualizes simple (single-variable) linear regression with generated synthetic data.
3. Very fast training speeds or long runs can reduce UI responsiveness on low-end hardware.
4. Most training settings are fixed in the config file instead of being fully adjustable in the UI.

6.2 Future Enhancements

We can implement the following methods into our application to further enhance its functionality:

1. Add UI controls to change learning rate, noise level, number of points, and max iterations during runtime.
2. Support importing real datasets (CSV) and exporting training results/metrics.
3. Extend to multivariate linear regression and show additional visualizations (e.g., parameter curves or loss surface view).
4. Include more optimization methods and features like early stopping and automatic convergence detection.

Source Code:

Linear Regression