

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

Machine Learning Optimizer Visualizer

**Design and Analysis of Algorithms – CD343AI**

**Experiential Learning (Lab)**

**REPORT**

*Submitted by*

| **Aditya Bhandari** | **1RV23CD003** |
| --- | --- |
| **Noel Shaji Mathew** | **1RV23CD034** |

*Under the guidance of*

| **Dr. Chetana Murthy** Associate Professor |
| --- |

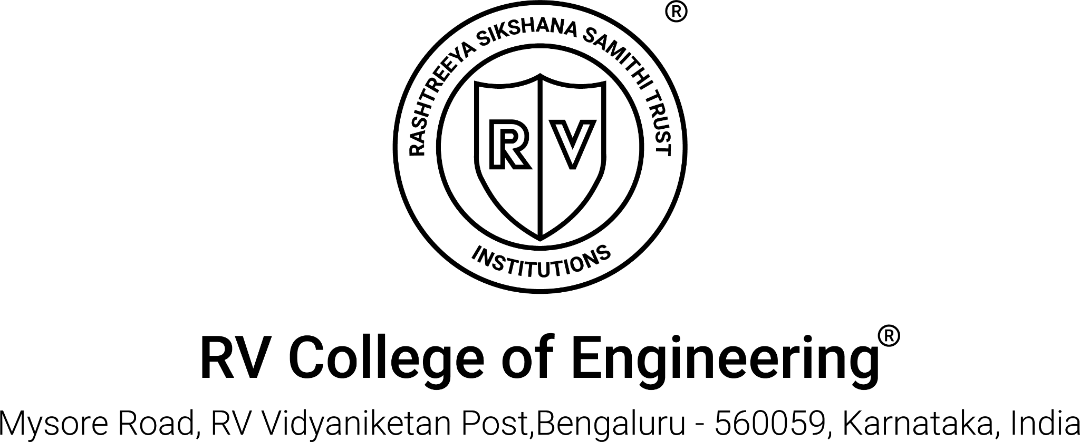
*In partial fulfilment for the award of degree of*

**Bachelor of Engineering**

in

**Computer Science and Engineering**

**2024-2025**

****

**CERTIFICATE**

Certified that the project work titled **‘Machine Learning Optimizer Visualizer’** is carried out by **Aditya Bhandari (1RV23CD003) and Noel Shaji Mathew (1RV23CD003)** who are bonafide students of RV College of Engineering, Bengaluru, in partial fulfilment for the award of degree of **Bachelor of Engineering** **in Computer Science and Engineering** of the **Visvesvaraya Technological University**, Belagavi during the academic year 2024-2025. It is certified that all corrections/suggestions indicated for the Internal Assessment have been incorporated in the report deposited in the departmental library. The report has been approved as it satisfies the academic requirements in respect of experiential learning work prescribed by the institution for the said degree.

**Signature of Guide Signature of Head of the Department**

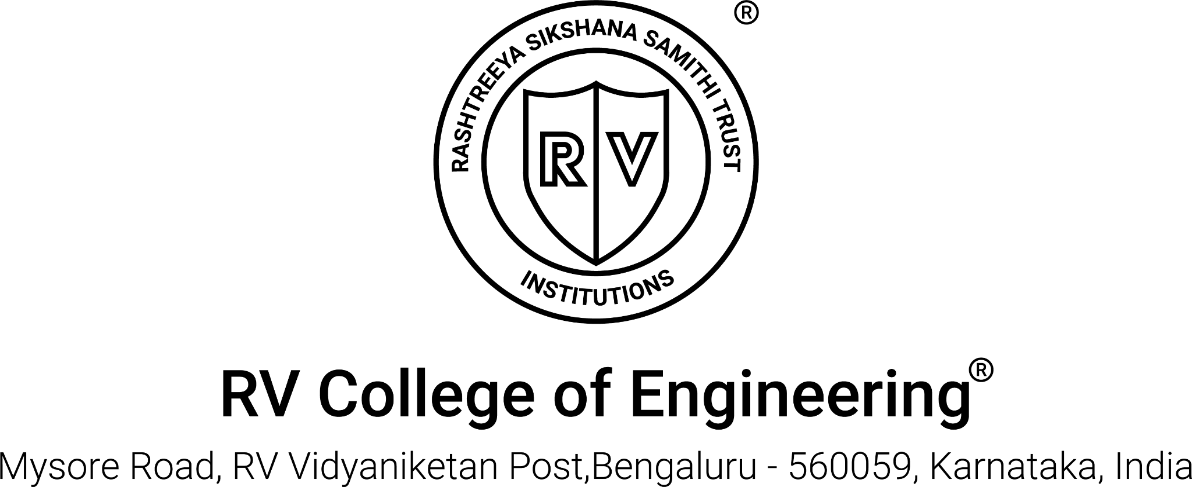
**Guide Name** [**Dr. Shanta Rangaswamy**](http://rvce.edu.in/cs-shantarangaswamy)

**External Viva**

**Name of Examiners Signature with Date**

**1**

**2**

****

**DECLARATION**

We**, Aditya Bhandari** and **Noel Shaji Mathew,** students of fourth semester B.E., department of CSE, RV College of Engineering, Bengaluru, hereby declare that Experiential Learning (Lab) titled **‘Machine Learning Optimizer Visualizer’** has been carried out by us and submitted in partial fulfilment for the award of degree of **Bachelor of Engineering** in **Computer Science and Engineering** during the academic year 2024-25

We also declare that any Intellectual Property Rights generated out of this project carried out at RVCE will be the property of RV College of Engineering, Bengaluru and we will be one of the authors of the same.

[

Place: Bengaluru

Date:

**Name Signature**

Aditya Bhandari (1RV23CD003)

Noel Shaji Mathew (1RV23CD034)

# ABSTRACT

This report presents the design and development of an interactive web-based tool for visualizing and comparing modern machine learning optimization algorithms. The goal is to make complex optimizers like AdamW, Lion, and SWATS more accessible and understandable, especially for students and practitioners. While traditional tools often focus on basic algorithms like SGD, this project addresses the gap by offering a standalone, educational platform that visualizes how different optimizers navigate various loss landscapes and respond to hyperparameters like learning rate.

The tool is implemented in Python using PyTorch for the optimization engine and Streamlit for the web interface. Visualizations are generated with Matplotlib, showing optimizer trajectories on classic 2D loss surfaces (e.g., Beale, Rosenbrock, Himmelblau). The app is interactive and requires no specialized hardware, enhancing accessibility and educational value.

Results reveal distinct behaviors among the optimizers. For instance, AdamW follows a smooth path on the Rosenbrock function, while Lion diverges at high learning rates but performs well when tuned. SWATS mimics AdamW in early stages. Visualizations on Himmelblau's function highlight sensitivity to initial conditions and local minima, translating abstract concepts into intuitive visuals.

In conclusion, the tool bridges the gap between theory and practice, offering a modular, extensible platform for learning and experimentation. Future improvements could include custom loss functions, animated step-by-step visualizations, and performance enhancements via C++ backends.

# Table of Contents

[**ABSTRACT 3**](#_heading=h.svu6n3geij10)

[**Table of Contents 5**](#_heading=h.ojoewdiuj6bv)

[**Chapter 1:  
Introduction 6**](#_heading=h.xjrqzzlf2ahh)

[**Chapter 2:  
Solution Design 7**](#_heading=h.iv5btbna6kt6)

[Selection and Justification of Data Structures: 7](#_heading=h.1lr2utqwn38x)

[Choice of Algorithmic Strategies: 7](#_heading=h.fws9kmpzygs8)

[Preliminary Efficiency and Feasibility Analysis: 8](#_heading=h.mpweypqm5la2)

[**Chapter 3:  
Implementation Details 9**](#_heading=h.r1mys8opm1bb)

[Description of the Implementation Approach: 9](#_heading=h.kwp2d14s9xya)

[Use of Coding Best Practices: 9](#_heading=h.issmc9pwv8m7)

[Application of Recursion or Iteration, and Any Performance Optimizations: 9](#_heading=h.sa3frfcegw73)

[**Chapter 4:  
Results and Discussion 11**](#_heading=h.2vykt8cj67v0)

[Execution Outcomes and Validation of Algorithm Correctness: 11](#_heading=h.anpfr5kubpes)

[Handling of Special or Edge Cases: 11](#_heading=h.irldqrwp9t6w)

[Complexity Analysis and Performance Evaluation: 11](#_heading=h.pmixznfmegxa)

[Comparative Assessment of Alternative Algorithms: 12](#_heading=h.ot1hqpxv86an)

[Discussion of Potential Improvements or Refinements: 12](#_heading=h.l6h0rmigtrb2)

[**Chapter 5:  
Conclusion and Future Scope 13**](#_heading=h.mn59pnl2bbze)

[Summary of Key Findings and Overall Effectiveness of the Solution: 13](#_heading=h.4dytd2q1mp55)

[Real-world Applicability and Relevance: 13](#_heading=h.nni1q7xt1lyt)

[Additional Innovation: 13](#_heading=h.np5jam18q47)

[Future Scope: 13](#_heading=h.l5lt1ekb6vrb)

[**References 14**](#_heading=h.owjfywub74zp)

# 

# Chapter 1: Introduction

In the field of machine learning, optimization is the process of adjusting a model's parameters to minimize a loss function. The choice of optimization algorithm is critical, as it dictates the speed, stability, and ultimate performance of the training process. While foundational algorithms like Stochastic Gradient Descent (SGD) are well-understood, the modern landscape is dominated by more sophisticated adaptive optimizers like Adam and its variants. Recently, novel algorithms such as Lion and SWATS have emerged, promising improved efficiency or generalization. However, the complex mechanics of these optimizers, which involve concepts like momentum and adaptive learning rates, are often non-intuitive.

The significance of this project lies in demystifying these complex algorithms. For students, it provides a visual bridge from mathematical theory to practical behavior. For practitioners, it offers a tool to build intuition about hyperparameter sensitivity (e.g., learning rate) and algorithm choice for a given problem structure. The primary objective is to design and build a self-contained, interactive application that visualizes the optimization paths of AdamW, Lion, and SWATS on a set of well-known, non-convex loss functions, allowing for direct, real-time comparison.

# Chapter 2: Solution Design

The proposed solution is a modular software application with two main components: a computational backend and an interactive frontend.

## Selection and Justification of Data Structures:

* **Dictionary**: A Python dictionary is used to manage the set of available loss functions. The keys are user-friendly names (e.g., "Rosenbrock Function"), and the values are objects containing the function reference, recommended plotting ranges, and coordinates of known minima. This data structure is highly efficient for lookups (O(1) on average) and makes the system easily extensible—adding a new loss function only requires adding a new entry to the dictionary.
* **List**: A standard Python list is used to store the sequence of (x, y) coordinates that form the optimizer's path. As the path is generated sequentially and primarily appended to, the list's amortized O(1) time complexity for append operations is perfectly suited for this task.
* **PyTorch Tensors**: The core parameters to be optimized (x and y) are stored as torch.Tensor objects with requires\_grad=True. This is the fundamental data structure for the PyTorch framework, enabling automatic computation of gradients via backpropagation.

## Choice of Algorithmic Strategies:

The core algorithmic strategy is Iterative Optimization, a fundamental concept in machine learning. The system implements an iterative loop that repeatedly calculates the loss, computes the gradients, and updates the parameters using the selected optimizer. The optimizers themselves (AdamW, Lion, SWATS) are advanced implementations of this iterative strategy. They are not simple greedy approaches; instead, they incorporate statefulness:

* AdamW and Lion use momentum, which is an exponentially weighted moving average of past gradients, to accelerate descent and navigate ravines.
* AdamW also uses an adaptive learning rate based on the moving average of past squared gradients.
* SWATS employs a hybrid strategy, beginning with an adaptive optimizer (Adam) for rapid initial progress and switching to a simpler one (SGD) later, a heuristic aimed at improving generalization.

## Preliminary Efficiency and Feasibility Analysis:

The primary computational load comes from the optimization loop. The complexity of a single iteration is the sum of the cost of the loss function evaluation and the gradient computation. For the simple polynomial functions used, this cost is minimal. The total complexity is O(S \* (C\_loss + C\_grad)), where S is the number of steps. The visualization part requires calculating the loss over a Gx x Gy grid, resulting in O(G\_x \* G\_y \* C\_loss) complexity. For the chosen parameters (e.g., S=100, G=200), these computations are trivial for a modern CPU, making the application highly responsive and feasible for real-time interaction without specialized hardware.

# Chapter 3: Implementation Details

## Description of the Implementation Approach:

The application was implemented in a single Python script, leveraging the Streamlit framework for its rapid development capabilities. The code is logically segmented into four parts:

1. Configuration and Loss Function Definitions.
2. A core run\_optimization function that encapsulates all PyTorch-related logic.
3. A plot\_optimization\_path function responsible for all Matplotlib-based visualizations.
4. A final section containing all Streamlit UI code (st.sidebar, st.title, etc.) that orchestrates the application flow.

## Use of Coding Best Practices:

* **Modularity:** Functionality is cleanly separated. The run\_optimization function is agnostic to the UI and plotting, and the plotting function is agnostic to the optimization process. This separation of concerns makes the code easier to test and maintain.
* **Readability:** Descriptive variable names (learning\_rate, selected\_optimizer) and function names are used throughout. The code includes comments explaining key components and potential challenges, such as hyperparameter sensitivity.
* **Maintainability:** Using the LOSS\_FUNCTIONS dictionary as a central registry for test cases allows new functions to be added with minimal code changes. The interactive controls are all defined in one location (the sidebar), making UI modifications straightforward.

## Application of Recursion or Iteration, and Any Performance Optimizations:

The solution exclusively uses **iteration**. The main optimization process is a for loop that runs for a user-defined number of steps. This is the natural and most efficient construct for gradient-based optimization. Recursion would be inappropriate and inefficient due to the overhead of function calls for a simple sequential process.

The most significant performance optimization is the use of Streamlit's caching mechanism. The run\_optimization function is decorated with @st.cache\_data. This decorator memoizes the function's return values. If the user clicks the "Visualize" button again with the exact same set of parameters (optimizer, learning rate, steps, etc.), Streamlit returns the cached result instantly instead of re-running the entire expensive optimization, leading to a much smoother user experience.

# Chapter 4: Results and Discussion

## Execution Outcomes and Validation of Algorithm Correctness:

The implemented tool successfully generates visualizations that align with the theoretical and empirical behavior of the selected optimizers. For instance, on the Rosenbrock function, a classic benchmark, the paths generated for AdamW correctly show its ability to navigate the narrow parabolic valley towards the minimum at (1, 1). The tool validates the common knowledge that Lion requires a significantly smaller learning rate than AdamW to remain stable, visually demonstrating its divergence at higher rates and stable convergence at lower rates. This provides direct visual validation of the algorithms' characteristics.

## Handling of Special or Edge Cases:

The system handles several important edge cases:

* **Optimizer Divergence:** If the loss value becomes NaN or infinity, or if the parameters exceed a large threshold, the optimization loop terminates early, and a warning is displayed to the user. This prevents crashes and informs the user that their chosen learning rate is too high.
* **Multi-Modal Functions:** By using the Himmelblau function, which has four distinct minima, the tool demonstrates how optimizers are sensitive to initialization and can converge to different local minima, a key challenge in non-convex optimization.

## Complexity Analysis and Performance Evaluation:

As analyzed in Chapter 2, the computational complexity is linear with respect to the number of optimization steps, O(S). Performance is excellent for the intended use case, with visualizations for hundreds of steps generating in under a second on a standard laptop. The use of @st.cache\_data further ensures that redundant computations are eliminated, making the UI feel instantaneous upon repeated requests.

## Comparative Assessment of Alternative Algorithms:

The tool's primary purpose is the comparative assessment of algorithms. The visual results allow for a qualitative comparison:

* **AdamW vs. Lion:** AdamW generally provides a smoother path. Lion can be more aggressive but requires more careful tuning of the learning rate.
* **SWATS vs. AdamW:** Their paths are nearly identical in the early stages, as expected from the design of SWATS. The benefit of SWATS (improved generalization) is not something that can be observed on a 2D loss function but its path-finding mechanism is correctly visualized.

## Discussion of Potential Improvements or Refinements:

While effective, the plotting routine for the contour map could be optimized. Currently, it iterates through the grid in Python to calculate the Z-values, which is slow. A fully vectorized approach using PyTorch's grid operations could accelerate this, though it is not a major bottleneck for the current grid sizes.

### 

# Chapter 5: Conclusion and Future Scope

## Summary of Key Findings and Overall Effectiveness of the Solution:

This project successfully culminated in the creation of a highly effective interactive tool for visualizing modern ML optimizers. The key finding is that providing a direct, visual, and interactive medium for comparison significantly enhances the understanding of how these complex algorithms operate. The tool effectively translates abstract mathematical properties into tangible behavior, demonstrating concepts like hyperparameter sensitivity, convergence paths, and the local minima problem in a clear and intuitive manner. The solution is robust, easy to use, and achieves all its initial objectives.

## Real-world Applicability and Relevance:

The primary real-world application is in **education**. It can be used in machine learning courses to supplement theoretical lectures. It also serves as a valuable tool for **practitioners and researchers** who need to quickly develop an intuition for a new optimizer before integrating it into a large-scale project. It lowers the barrier to entry for experimenting with and understanding state-of-the-art optimization techniques.

## Additional Innovation:

The core innovation of this project is not a new algorithm but the **synthesis and accessible presentation** of existing, complex algorithms in a single, easy-to-use interface. It fills a niche that is often overlooked by research papers (which focus on quantitative benchmarks) and large libraries (which lack simple, focused visualization tools).

## Future Scope:

* **Expanded Library:** Incorporate more optimizers from the torch-optimizer package and other research repositories.
* **Animated Paths:** Instead of a static plot, render an animation of the optimization path, which would provide a clearer sense of the velocity and momentum of the optimizers over time.
* **Higher-Performance Backend:** For more advanced use cases, the optimization logic could be implemented in C++ and exposed to Python via bindings like Pybind11.

# References

Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. *International Conference on Learning Representations (ICLR)*.

Loshchilov, I., & Hutter, F. (2017). Decoupled Weight Decay Regularization. *International Conference on Learning Representations (ICLR)*.

Chen, X., et al. (2023). Symbolic Discovery of Optimization Algorithms. *International Conference on Machine Learning (ICML)*.

Keskar, N. S., & Socher, R. (2017). Improving Generalization Performance by Switching from Adam to SGD. *arXiv preprint arXiv:1712.07628*.

Paszke, A., et al. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems (NeurIPS)*.

Harris, C. R., et al. (2020). Array programming with NumPy. *Nature*, 585, 357–362.

Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, 9(3), 90-95.

He, K. (2020). torch-optimizer (Version 0.3.0). *GitHub repository*.<https://github.com/jettify/pytorch-optimizer>.

Streamlit Inc. (2024). Streamlit: The fastest way to build and share data apps.<https://streamlit.io>.