The National Institute of Engineering, Mysuru
Department of Computer Science & Engineering (AIML)

GLACIER.ML

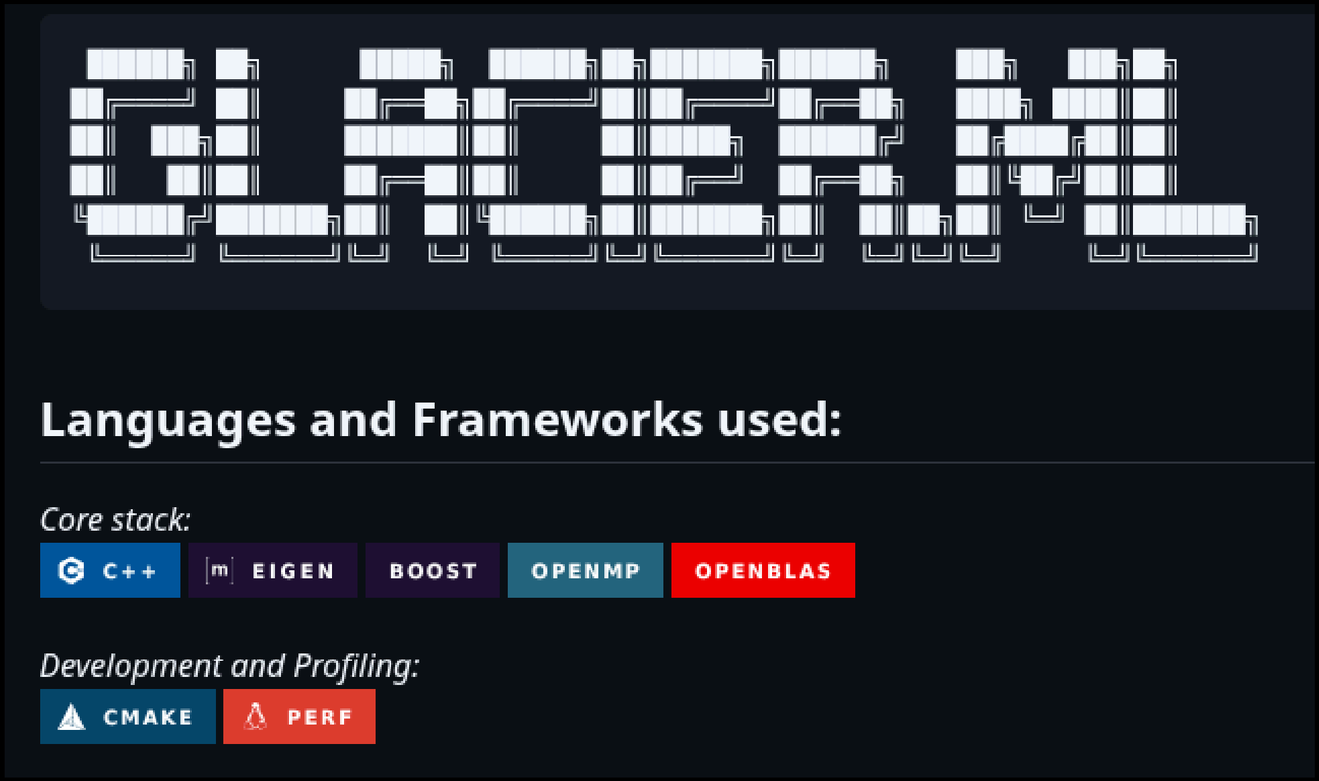
MINOR PROJECT EVALUATION - PHASE 2

NOVEMBER 13, 2025
Batch - F11

Project guide: Ms. Mahe Mubeen Aktar D
Team: Skandan C.Y (4NI23CI103)

TABLE OF CONTENTS

01. INTRODUCTION



Glacier.ML is a header only Supervised Machine Learning library, built entirely using C++.

Aims to be a lightweight, fast alternative to **Scikit-learn**

Currently, it includes seven models, with four more in development. Benchmarking shows Glacier.ML consistently outperforms Scikit-learn models in execution speed, while maintaining comparable predictive accuracy.

Link: https://github.com/skandanyal/Glacier.ML

02. LITERATURE SURVEY

Sl no.	Title	Authors	Year of publication
1.	Optimization Of KNN, SVM, And SVM Kernel in Water Potability Prediction with Hyperparameter Approach	Roy Hendro Siburian et al.	2025
2.	Ensemble-based Machine Learning and Deep Learning Approaches for Autism Prediction	Dr.K.Suthendran et al.	2025
3.	Simulation of Hierarchical Parallel Computing Model for Fluid Machinery Based on Support Vector Machines	Dechen Wei et al.	2023
4.	Design and Implementation of Parallel Processing Algorithm for Big Data in High Performance Computing Framework	Qihao Dong et al.	2023
5.	Comparative Analysis Of Optimization Techniques On Multi-class SVM	Neha J Deshpande et al.	2022
6.	Performance portable Vlasov code with C++ parallel algorithm	Yuuichi Asahi et al.	2022

OBSERVATIONS:

- Modern HPC applications are being abstracted by Python.
- Could not find papers on optimizing Supervised ML algorithms in the last 5 years on IEEE Xtreme.
- Algorithmic optimization trends are appearing more in applications in edge devices

03. EXISTING SYSTEM, FORMULATION OF PROPOSED SYSTEM

EXISTING SOLUTION - SCIKIT-LEARN

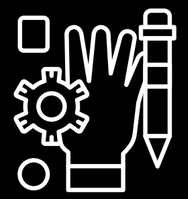
A Production-Grade Pythonic API:

- **High level**, **industry grade** Python API, used extensively throughout the **world.**
- Contains robust and easy documentation, enabling fast prototyping and useage.

Highly-Optimized C, Cython Core:

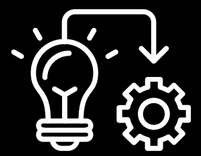
- Majorly uses NumPy and SciPy, which themselves are powered by C/C++, followed by Cython and C in performance critical regions and Joblib to implement parallelization.
- Code is highly optimized by numerous researchers and professionals over the years.

PROPOSED SYSTEM – GLACIER.ML



Self motivated hands-on initiative:

- Solo, systems-level high performance project to build core ML algorithms from scratch in C++.
- Aimed to gain deep, first-principles understanding of algorithmic design, memory management, and parallel computing through implementation.

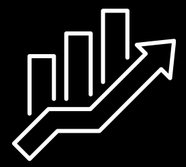


Insights through implementation:

- Uncovers the **how** and **why** behind implementation and resulting performance.
- Initial naive implementations serve as a baseline, revealing performance bottlenecks and room for optimizations.

Solo developer team:

- Allows me to play multiple roles required in an end-to-end software project, giving me invaluable hands-on experience.
- Allows me to use important professional tools and practices, which a conventional project would not provide.

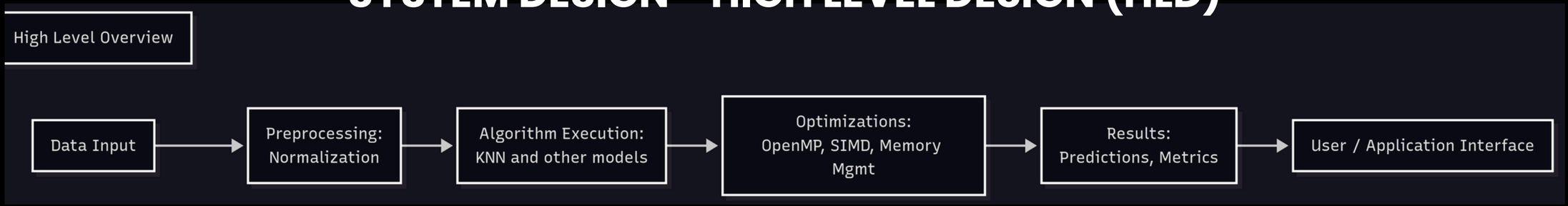


From Naivete to Performance:

- KNN, once **200x** slower than Scikit-learn, now runs only about **4x** slower, while SVM achieves **4-10x** faster execution.
- These performance gains stem from extensive **OpenMP** parallelization and **OpenBLAS** backed linear algebra acceleration.

04. SYSTEM DESIGN

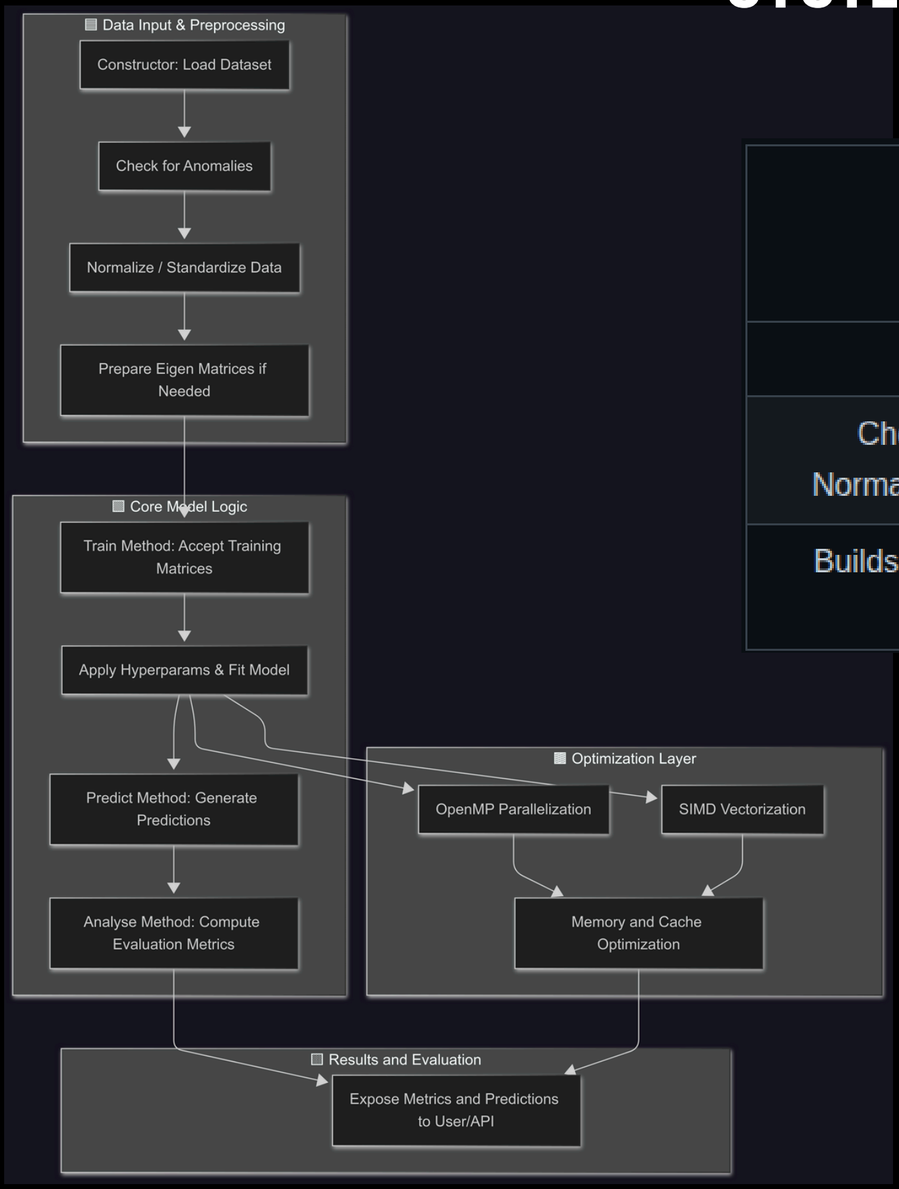
SYSTEM DESIGN – HIGH LEVEL DESIGN (HLD)



Architecture overview
Modular framework structure
Core ML algorithms (KNN, SVM, extensible for future models)
Optimization layer (OpenMP, SIMD, memory management)
Data handling layer (loading, preprocessing, normalization)
Abstraction boundaries for clean extensibility
Separation of algorithm logic vs. optimization logic
Designed for future scalability

Workflow overview
Input data ingestion
Preprocessing (cleaning, normalization, transformation)
Algorithm selection (KNN, Logistic Regression, etc.)
Model execution (distance computation, kernel evaluation, etc.)
Parallelization & vectorization applied at critical steps
Results aggregated into predictions
Output delivered (predictions, performance metrics)

SYSTEM DESIGN – LOW LEVEL DESIGN (LLD)



Data input & Pre-processing `constructor()`
Constructor loads data
Checks for anomalies in the dataset, Normalizes the data for easier computation
Builds the Eigen matrices for mathematical computation

Core Model Logic `.train()`
Accepts hyper-parameters
Conducts model specific mathematical computations
Stores parameters and hyprer-parameters for predicting the results

Optimization layer `constructor()` & `.train()`
Uses cache-friendly flat matrices for memory optimization
Uses OpenMP to implement multithreading and SIMD instructions (vectorization)

Results and Evaluation `.predict()` & `.analyze()`
Produces the result using stored parameters and hyper-parameters
Generates task specific evaluation metrics

05. SYSTEM REQUIREMENTS

HARDWARE & SOFTWARE REQUIREMENTS

Hardware	requirements
Processor	Any multi-core CPU Intel i5 / AMD Ryzen 5 or better
Memory	Minimum 4 GB RAM, 8 GB recommended
Any other device	Standard PC / Laptop

Tools	Purpose
Version control	Git and GitHub
Build System	Cmake
IDE	CLion and PyCharm
Server	Golang with Gin or Echo framework
Profiling	Perf
Testing	GTest

Purpose	Language and libraries
Core logic	C++ 20, Eigen, Boost, OpenMP, OpenBLAS
Website infrastructure	Golang, HTML and CSS, Javascript, Javascript
Database	PostgreSQL
Benchmarking	Python, Numpy, Pandas, Scikit-learn, Matplotlib, Seaborn

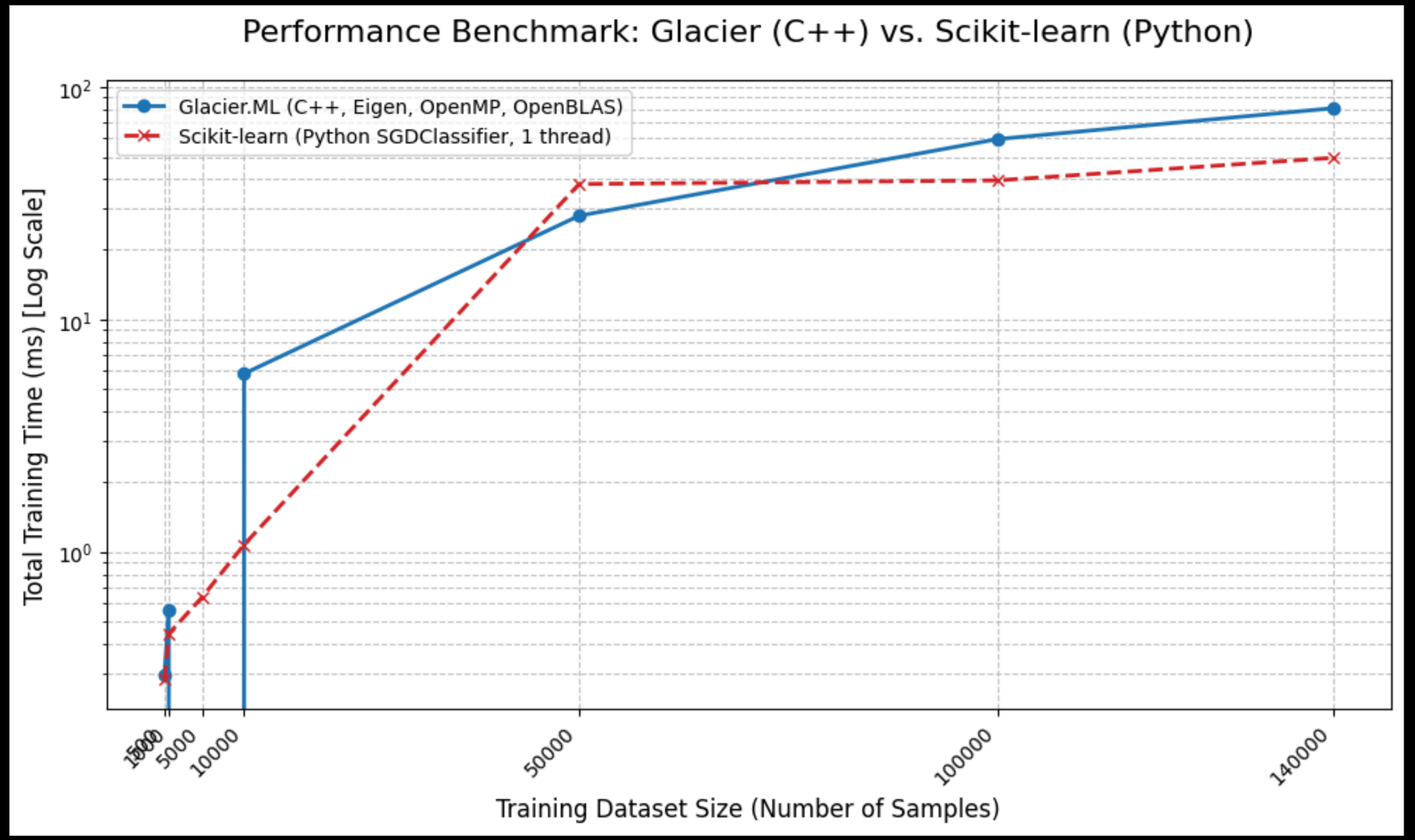
OS	Purpose
Linux (Ubuntu 24.0)	For development
Linux / Windows / MacOS	For useage

06. IMPLEMENTATION

- Built entirely in **C++20** as a **header-only library**, integrated via **CMake**.
- Core math powered by **Eigen** and **OpenBLAS** for optimized linear algebra.
- **OpenMP** ensures multi-core parallelism; **SIMD** accelerates compute-heavy loops.
- Performance profiling and tuning handled through **perf**, guiding cache and vectorization optimizations.
- All models implemented from scratch, enabling full control over efficiency and memory layout.
- Mathematical intuition behind the algorithms was developed using resources from **Stanford Online's Statistical Learning (Python)** course on YouTube.

07. RESULTS - CLASSIFICATION MODELS

07a. Logistic Regression



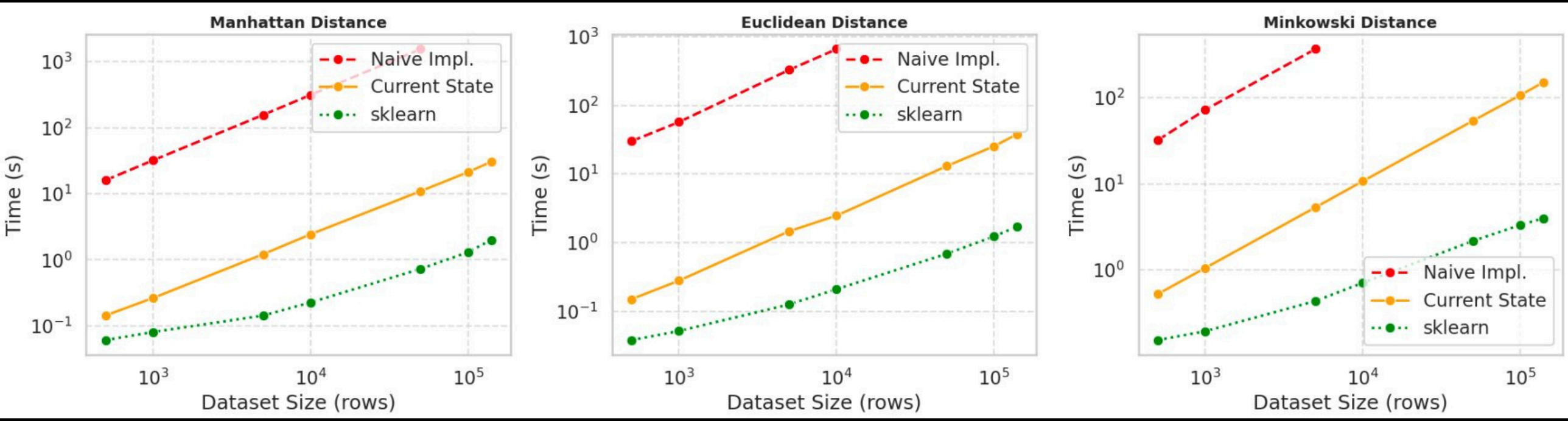
Observation:

Mixed results depending upon the size of the dataset.

NOTE:

Models were benchmarked on an **AMD Ryzen 5 6600H** SoC with **6 cores** and **12 threads.**

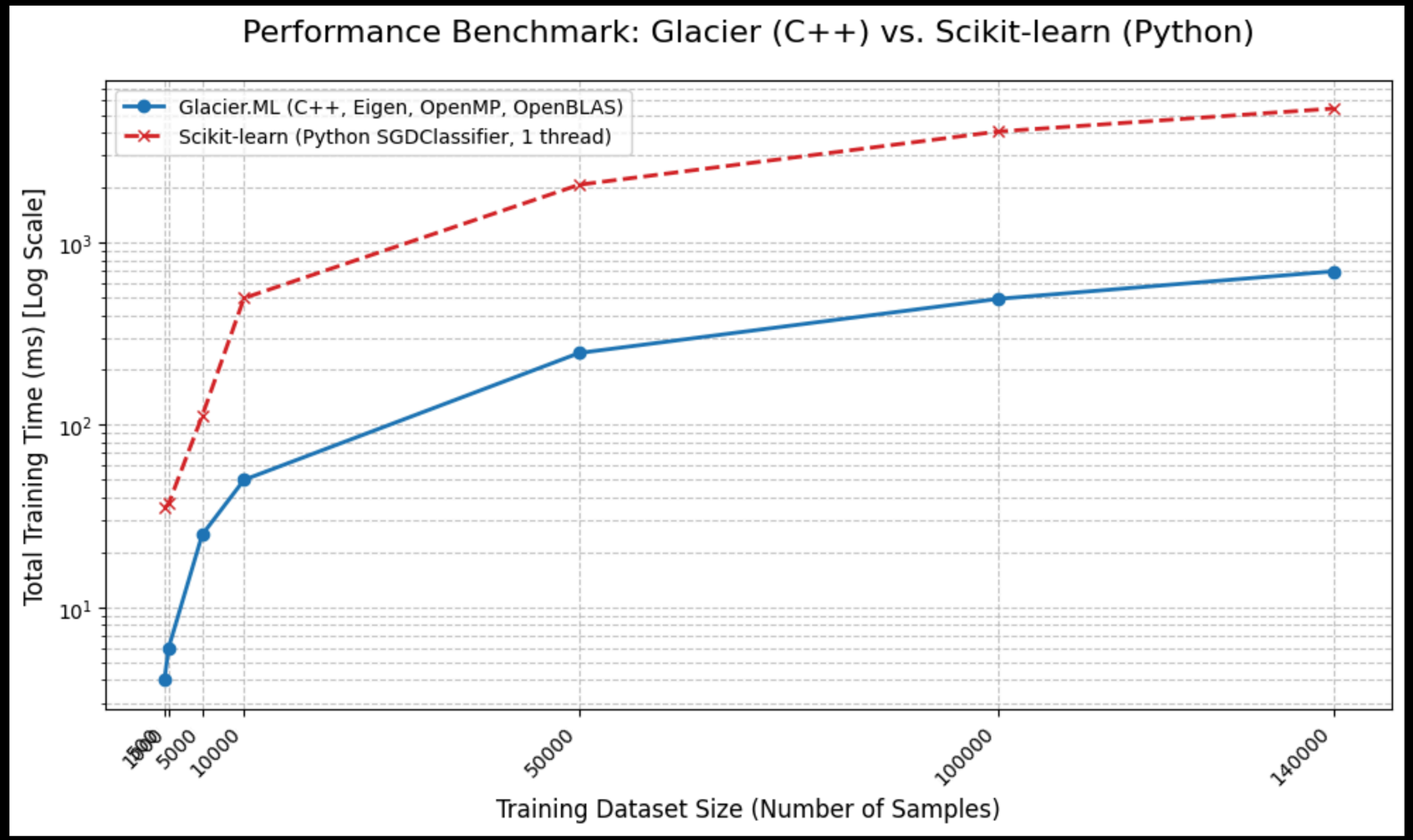
07b. KNN Classifier



Observation:

- **100-250x** faster than naive implementation
- **4-6x** slower than Scikit-learn equivalent

07c. SVM Classifier



Observation:

- Consistently **4-10x** faster than closest model from Scikit-learn.

 # There exists no directly equivalent model from Scikit-learn. Model here has been benchmarked against SGDClassifier.

08. FUTURE ENHANCEMENTS & CONCLUSION

FUTURE ENHANCEMENTS

- Open-source Glacier.ML after the minor project phase to enable community contributions.
- Add **GPU** acceleration once the CPU baseline stabilizes.
- Improve robustness, error handling, and null-value support similar to Scikit-learn.
- Yet to be tested on systems with varying hardware specifications.

CONCLUSION

- Glacier.ML is nearing Scikit-learn in performance.
- Achieves **600–1000x** better memory efficiency.
- Scikit-learn remains more robust and fault-tolerant.
- Currently a solo-developed project; aims to expand into a collaborative effort.
- Future focus: **stability, scalability,** and **GPU integration.**

09. REFERENCES

- **Statistical Modelling with Python** by **Stanford Online:** To understand the various mathematical models to implement.
- **“Complete System Design Roadmap 2025” by Apna College:** To understand the concept of system design and it’s importance.
- **Scikit-learn repository:** To understand the underlying code structure.
- **Google Scholar, ResearchGate, arXiv, IEEE Xplore:** To gather relevant research papers.

THANK YOU!