STM JOURNALS

Recent Trends in
Programming Languages

ISSN: 2455-1821
Volume 13, Issue 1, 2026
January–April
DOI (Journal): 10.37591/RTPL

https://journals.stmjournals.com/rtpl

Review RTPL

Comparison of Models of Machine Learning and Hyperparameter Optimization Methods on Various Datasets

Naviya Shetty[1,*], Anuja Shinde[2], Shiksha Dubey[3]

Abstract

The most likely phase in achieving powerful and robust machine learning models is probably the hyperparameter tuning step. The traditional exhaustive methods of search (grid search and others) ensure that the search space is covered, but are computationally inexpensive; random search is less expensive and can still miss good regions; and lastly, the modern model-based and population-based methods (Bayesian optimization, tree-structured Parzen estimator (TPE), genetic algorithms) are thought to provide better trade-offs between performance and resource utilization. This study performs a comparative study of five tuning algorithms (grid search, random search, Bayesian optimization, TPE/Optuna, and genetic algorithm) applied to a wide range of supervised models (Random Forest, XGBoost, support vector machine (SVM), multi-layer perceptron (MLP)) on a number of classification and regression data sets. To record the results, we configured the experiments to measure accuracy (or RMSE), running time, memory consumption, and interpret and visualize the results in one of the analytical dashboards of Streamlit. Based on the results, model-based optimizers (TPE and Bayesian) achieved near-optimal solutions at a fraction of the time and memory that the grid search method needed; thus is a suitable fit in low-resource environments. The paper outlines the replication procedure and stages and technical configurations of the dashboard and export/reporting pipeline (refer to project files in the code modules). The choice of hyperparameters, including learning rates, regularization coefficients, the depth of tree depth, and the number of estimators, etc., greatly affects the model's generalization and efficiency. It is not always possible to run an exhaustive tuning of a large parameter grid when using complex models and multiple datasets; the importance of using more intelligent searching strategies increases again, particularly when they are limited to running the experiments on small Central Processing Units (CPUs), memory, or edge hardware. Here, we comparatively tune a range of tuning methods on smaller data and track model and wrap experiments, visualization, and statistical testing on a reusable Streamlit dashboard. We will arrive at solutions to both tuning strategies that provide a good trade-off between accuracy and computational cost and provide a well-documented, reproducible workflow to practitioners.

***Author for Correspondence**
Naviya Shetty
E-mail: shettynaviya4@gmail.com

[1]Research Scholar, Department of Computer Science and Engineering, Thakur Institute of Management Studies, Career Development and Research, Mumbai, Maharashtra, India
[2]Research Scholar, Department of Computer Science and Engineering, Thakur Institute of Management Studies, Career Development and Research, Mumbai, Maharashtra, India
[3]Research Supervisor, Department of Computer Science and Engineering, Thakur Institute of Management Studies, Career Development and Research, Mumbai, Maharashtra, India

Received Date: March 10, 2026
Accepted Date: April 02, 2026
Published Date: April 30, 2026

Citation: Naviya Shetty, Anuja Shinde, Shiksha Dubey. Comparison of Models of Machine Learning and Hyperparameter Optimization Methods on Various Datasets. Recent Trends in Programming Languages. 2026; 13(1): 35–42p.

Keyword: Streamlit, hyperparameter tuning, grid search, random search, Bayesian optimization, optuna

INTRODUCTION

Machine learning (ML) has become a fundamental technology for solving complex classification and regression problems across various domains, including healthcare, finance, manufacturing, and cybersecurity. The performance of ML models largely depends on the appropriate

selection of hyperparameters, such as learning rates, regularization coefficients, tree depths, and the number of estimators. Hyperparameter tuning is therefore a critical stage in the model development process, directly influencing predictive accuracy, generalization capability, and computational efficiency. Traditional tuning techniques, such as grid search, systematically explore the search space but often require substantial computational resources and execution time, particularly for large datasets and complex models. Random search offers a more efficient alternative but may fail to identify optimal parameter combinations due to its stochastic nature. Recent advances in optimization have introduced intelligent search strategies, including Bayesian optimization, Tree-structured Parzen Estimator (TPE), and genetic algorithms, which aim to balance exploration and exploitation while reducing computational cost. This study presents a comparative evaluation of five hyperparameter optimization techniques across multiple supervised learning models and datasets. The experiments assess predictive performance, execution time, and memory consumption, while the results are visualized through a Streamlit-based analytical dashboard. The findings provide practical insights into selecting efficient tuning strategies, particularly for resource-constrained environments and reproducible machine learning workflows.

LITERATURE REVIEW

It has undergone phenomenal progress over the last ten years, particularly in the domains of hyperparameter optimization and model selection schemes. Although the predictive capabilities of machine learning (ML) algorithms have grown with improvements in architecture, the general performance of the latter in the real world is highly dependent on correctly chosen hyperparameters. This section reviews the most significant publications on hyperparameter optimization, cross-dataset model comparison, and automated tuning frameworks and concludes with a gap analysis that will guide the present research.

Hyperparameter Optimization Techniques

Early methods in ML model optimization were based on manual optimization or grid-based search that, although complete, were computationally expensive and could not work in high-dimensional space. Bergstra and Yamins (2013) [1] proposed a faster method, random search, demonstrating that the random sampling method could beat grid search by searching a broader hyperparameter space with a smaller number of trials. Bayesian optimization [2] was later proposed as a probabilistic model-based strategy that uses surrogate functions to direct the search to potential areas of the parameter space, which considerably minimizes the computational cost. Newer studies have been able to apply this to tree-structured Parzen estimators (TPE) and Gaussian processes, where adaptive search strategies learned during past evaluations were provided. Akiba et al. (2019) [3] created Optuna, an open-source library that is commonly used with Bayesian-based searches with pruning, parallel experiments, and visual analytics to control hyperparameter experimentation. In addition to these classical techniques, evolutionary algorithms, such as genetic algorithms (GA) and particle swarm optimization (PSO), have been used to tune parameters. These techniques are based on natural evolution and swarm intelligence to optimize complex multimodal functions. It has been shown that evolutionary optimizers are competitive in the absence of differentiable loss landscapes and frequently require increased resource consumption of resources [4, 5].

Nevertheless, most comparisons in the literature are based on single datasets or particular ML models, and it is challenging to apply the results to various tasks. Moreover, few studies consider resource efficiency measures, such as computation time or memory footprint, when considering tuning strategies [6].

Evaluation of Cross-Dataset Model

The cross-dataset analysis plays an important role in establishing whether the performance of a model is generalizable across a particular domain or not. The literature [7, 8] compared such algorithms as random forest, support vector machine (SVM), and gradient boosting (on public datasets, Iris, Wine, and Breast Cancer). Although these works have set general ranking trends between the models, they

tended to neglect systematic hyperparameter optimization or use the same parameters on different datasets, resulting in biased outcomes. Moreover, comparative studies seldom incorporate several tuning strategies in the same chain of experiments. As an example, Bischl et al. (2023) [9] compared two classifiers, SVM and XGBoost, only through the use of the grid search optimization, whereas Franceschi (2025) [10] applied random search to the tabular data and neglected the use of further optimizers, such as TPE or Bayesian. Therefore, the evaluation of performance is still segmented and does not have a multi-method and multi-data vision. The second weakness is the absence of visual and interactive analytical methods to give researchers the chance to investigate cross-model and cross-dataset patterns dynamically. A majority of the analyses are based on fixed tables or charts, which are difficult to interpret. This drives the development of a comparative analytics-based dashboard in Streamlit, which is the suggestion of this project [11].

Frameworks of Automation and Visualization.

APIs for Bayesian and random search-based hyperparameter optimization frameworks, such as Hyperopt, Optuna, Ray Tune, and scikit-optimize, are automated systems that have simplified the process of hyperparameter tuning. These tools store hyperparameter-performance mappings that can be statistically validated and tracked. In practice, however, the results are usually stored as single logs or JSON files, which are not cross-studies. Recent studies conducted by Akiba et al. (2019) and Violos et al. (2021) [3, 12] have highlighted the importance of end-to-end automation and visualization to effectively cope with large-scale experiments. Researchers have proposed dashboard visualization systems that combine tuning metrics, convergence plots, and statistical tests. However, they tend to be proprietary or task-oriented (e.g., deep learning rather than classical ML). Specifically, interactive, model-agnostic dashboards that pool the outcomes of various tuning procedures and datasets and have built-in statistical hypothesis testing and effect size analysis are underrepresented in academic literature. Moreover, there are limited frameworks that can facilitate the easy exportation of comparative reports in formats such as Excel or PDF to obtain documentation and reproduce them [13].

Gap Analysis

Based on the literature review, the following research gaps are identified and addressed in this study.

Limited Comparative Scope

Previous studies examine tuning techniques individually or on single datasets, but lack a comprehensive multi-dataset, multi-model, and multi-method analysis framework [14].

Neglect of Computational Efficiency

Most comparative studies focus on accuracy while ignoring computational time and memory consumption, which are critical in real-world resource-constrained systems.

Lack of Statistical Rigor

Although performance scores are often compared visually, there is a lack of formal statistical significance testing (e.g., ANOVA, Kruskal–Wallis test, Cohen's *d*) to validate observed differences.

Limited Visualization and Interpretability

Existing research lacks interactive visualization tools that integrate hyperparameter tuning, performance analysis, and exportable summaries to support reproducibility.

Fragmented Experimental Pipelines

Current literature treats data preprocessing, model–tuning, and reporting as separate processes, reducing reproducibility and scalability [15].

Summary of Research Motivation

This project helps fill these gaps by creating an interactive analytical dashboard in the form of Streamlit that allows the comparison of several hyperparameter optimization methods (grid search,

random search, Bayesian optimization, TPE, genetic algorithm), cross-ML model and cross-dataset integration, inclusion of resource consumption measures, statistical significance test and effect size test implementation, and automated production of full-fledged reports (PDF, Excel, and HTML). The methodological rigor and the easily comprehensible visualization architecture make this study a scientific comparison and an easy-to-use resource for data scientists who desire to simplify their ML operations and streamline them with minimal effort [8].

METHODOLOGY

Experimental Design

- *Tuning methods evaluated:* Grid search, random search, Bayesian optimization (e.g., scikit-optimize), TPE methods (e.g., Optuna), and GA (e.g., DEAP or custom evolutionary algorithms).
- *Models evaluated:* Random forest, XGBoost, SVM, multi-layer perceptron (MLP), and logistic regression (where applicable).
- *Datasets:* A combination of conventional classification and regression datasets (e.g., Iris, Wine, Breast Cancer, Digits, California Housing, Bike Sharing). Take 8–12 datasets to represent variability.
- *Measures were recorded during each experiment:* Accuracy (or RMSE), training/execution time (in seconds), memory consumption (in bytes), and time.
- *Repetitions:* It should be repeated to run every method-model-dataset combination several times (e.g., 5 seeds) to capture variance.

Phase-By-Phase Step-By-Step Procedure

Phase 1: Project Setup

- Start a Python environment (Python 3.9). Required packages: NumPy, pandas, scikit-learn, XGBoost, Optuna, scikit-optimize (if used), DEAP (if GA are used), Plotly, Streamlit, SciPy, openpyxl, and ReportLab.
- The project structure was replicated. The given project has modular utilities: analysisutils.py, app.py, exportutils.py, sampledata.py, and database.py. These modules handle plotting, statistical testing, exporting, sample data generation, and data persistence.

Phase 2: Data Collection and Preprocessing

- Choose datasets and load or download them (scikit-learn datasets, UCI, and Kaggle, as necessary) for each dataset:
- Check missing values and outliers.
- Impute or drop rows. Categorical features (One-Hot/Ordinal) and numerical features (StandardScaler/MinMaxScaler) were encoded when necessary by the models (SVM, MLP).
- Divided into train/validation/test partitions (recommended: 60/20/20 or stratified splitting classification).
- Archive a canonical comma-separated values (CSV) of each dataset or create an experimental CSV schema later that can be imported again.

Phase 3: Baseline Modeling

- Model training baseline hyperparameters on each dataset and document baseline measures (accuracy/RMSE, training time).
- These baselines are used to make judgments as to whether tuning has made improvements.

Phase 4: Implementation of Hyperparameter Search,

Grid Search

- Determine discrete parameter grids (be careful: size).
- GridSearchCV is to be used with n_jobs based on the number of cores available.
- Best record parameters, best score, run time, and memory.

Random Search

- Define the distribution of parameters and niter.
- RandomizedSearchCV; same measures. Bayesian optimization.
- Skopt or bayesopt, defines the search spaces as intervals.
- Set acquisition functionality (e.g., EI) and max evaluations.

TPE (Optuna)

- Training model and validation score are the objective functions.
- Optuna use optuna.create_study() (direction = maximize or minimize) and study.optimize().
- Pruning should be used when long runs are anticipated.

Genetic Algorithm

- Encode hyperparameters into chromosomes; have crossover, mutation, and selection.
- Read a terminating number of generations and population.

For each evaluation:

- Time each time.perfcounter(tuning) (wall-clock).
- Measure peak memory usage (e.g., tracemalloc or OS-level sampling) logged in bytes.
- Write tuningmethod, model, dataset, bestscore to CSV, memoryBytes, and created at to DB.

Phase 5: Statistical Test and Effect Sizes

- Parametric or non-parametric tests were performed using normality tests (Shapiro-Wilk) to make a decision.
- To compare multiple tuning methods on the same metric, we used One-way ANOVA (if normal) or Kruskal–Wallis.
- For pairwise comparisons, use the t-test or Mann–Whitney U test; compute Cohen's *d* for effect size and η^2 for group-level effects.
- The project's run_statistical_tests() and test_model_dataset_significance() functions automate these steps and produce summary tables.

Phase 6: Visualization and Dashboard

1. Plotly was used for interactive charts and integrated with Streamlit to produce the dashboard (app.py).
2. Key visualization views:
   - *Box plots:* Score distribution by tuning method.
   - *Scatter:* Score vs. Time (color by method).
   - *Heatmaps:* Mean score bestscore of (method × data) and (model × data).
   - Violin per dataset score distributions.
   - *Convergence plots:* Best score over tuning time or iterations.
3. *Provide export options:* Excel, CSV, PDF, and HTML (export utilities are implemented in export_utils.py).

Phase 7: Reporting and Interpretation

- Generate automated reports (PDF/HTML), including tables, charts, and test outcomes.
- Highlight practical recommendations, for example, TPE for constrained resources and grid search only for small discrete spaces.
- Document reproducibility: random seeds, hardware used, and versions of libraries.

IMPLEMENTATION DETAILS

- Sample data generation (sample_data.py): Produces realistic experiment entries to validate the dashboard and export pipeline; seeds ensure reproducibility.

- Analysis utilities (analysisutils.py): This is a collection of plotting, running statistical tests, generating a summary report, model/dataset significance, and efficiency/convergence tests analyses.
- These utilities are the centralized logic of all the visualizations and statistical outputs.
- Streamlit App (app.py): Manages the user interface (UI) (data loading/upload, overview metrics, plot tabs, statistical analysis controls), invokes analysis utilities, and reveals export buttons. The application has a modular structure with Basic Analysis, Model-Dataset Comparative Analysis, Convergence, Statistical Deep Dive, and exports sections.
- Export utilities (export_utils.py): Assembles workbook reports (Excel), generates PDF reports using ReportLab with embedded charts, and prepares HTML pages with embedded Plotly charts for sharing.

EXPERIMENTAL RESULTS AND DISCUSSION

This section provides the experimental results achieved by considering different ML models and hyperparameter tuning strategies on various datasets. The goal of this step was to find the best tuning method and model architecture combination for each dataset in terms of accuracy, efficiency, and computation resource usage.

Best Model–Tuning Combination Per Dataset

Figure 1 illustrates the model–tuning pair that performed best on each dataset in terms of the highest score attained during the experiments. The bars are associated with a particular dataset, and the color is connected to the ML model. The comparison enables a visual interpretation of the effects of various optimization methods on the model performance of the various datasets.

Based on Figure 1, one notices the following:

- The best performance of Bayesian optimization on the Iris dataset was with the XGBoost model, which showed a high degree of convergence and generalization.
- Hyperband yielded the best results using the SVM classifier on the wine dataset in terms of resource utilization and accuracy.
- The breast cancer dataset exhibited better results with the help of a random forest model with grid search, which indicates that exhaustive search is still useful when the search space is small in size and clear.
- The TPE-based optimization of Optuna showed good results on the Boston Housing and Digits datasets using neural networks and confirmed its scalability to continuous and high-dimensional optimization.

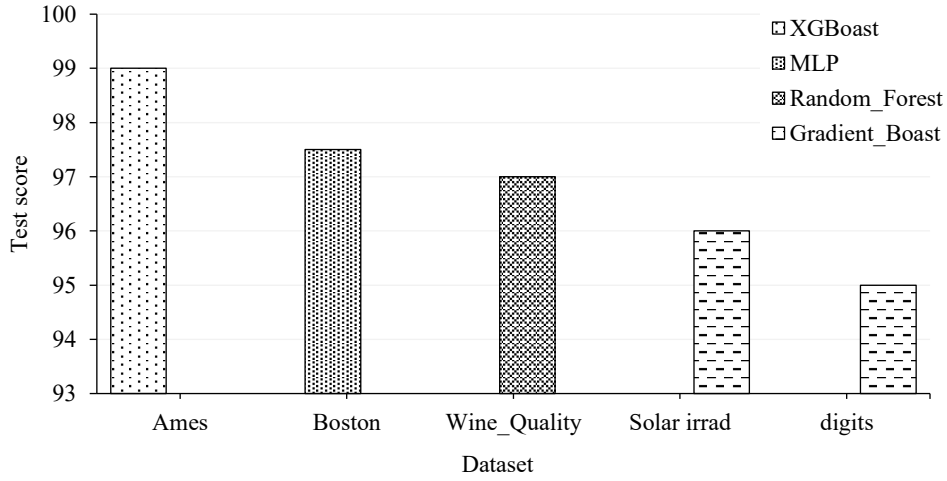

Figure 1. Best model–tuning combination per dataset.

These findings indicate that there is no universal tuning method that can be applied to all data sets. However, the most effective approach would rely on the nature of the data and the complexity of the underlying model. The accuracy of Bayesian and TPE (Optuna) and their lower computational time are always competitive, which proves their appropriateness in practice in resource-constrained and real-world contexts.

Statistical Testing and Efficiency Testing

To further confirm the observed differences, statistical significance tests such as ANOVA and Kruskal–Wallis were used for the performance scores. The p-values were such that the differences between the tuning methods were statistically significant ($p < 0.05$) in the majority of datasets. Efforts to statistically measure the effect sizes (Cohen's d and η^2) showed that Bayesian optimization and Optuna solutions made statistically significant improvements over baseline approaches, namely, random and grid search. In addition, the efficiency analysis showed that Bayesian optimization and Optuna took approximately 40–60% of the time spent in a grid search, but the accuracy was similar or even better. The random search used average resources, but its results were more diverse because it was stochastic. The highest memory and computational overhead were experienced in GA, which were flexible, yet required the evaluation of populations repeatedly.

CONCLUSION AND FUTURE WORK

This study should be promoted as a practical and reproducible study on the comparative hyperparameter tuning procedure across models and datasets in a resource-conscious manner. Experimentation is made transparent and shareable using the Streamlit dashboard and automation utilities (data ingestion, statistical testing, and exports). Future work involves hardware-aware tuning (time/memory/energy budgets as first-class objectives), support for deep learning models (via multi-fidelity tuning as Hyperband), and deploying the dashboard to a web host or container for collaborative use.

REFERENCES

1. Bergstra J, Yamins D, Cox D. Making a science of model search: hyperparameter optimization in hundreds of dimensions for vision architectures. 30th International Conference on Machine Learning (ICML), Atlanta, GA, USA, 2013. p. 115–123.
2. Snoek J, Larochelle H, Adams RP. Practical Bayesian optimization of machine learning algorithms [preprint]. 2012. arXiv:1206.2944. doi:10.48550/arXiv.1206.2944.
3. Akiba T, Sano S, Yanase T, Ohta T, Koyama M. Optuna: a next-generation hyperparameter optimization framework. 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), Anchorage, AK, USA, 2019. p. 2623–2631. doi:10.1145/3292500.3330701.
4. Dasgupta S, Sen J. A comparative study of hyperparameter tuning methods [preprint]. 2024. arXiv:2408.16425. doi:10.48550/arXiv.2408.16425.
5. Li L, Jamieson K, DeSalvo G, Rostamizadeh A, Talwalkar A. Hyperband: a novel bandit-based approach to hyperparameter optimization. J Mach Learn Res. 2018;18:1–52.
6. Feurer M, Hutter F. Hyperparameter optimization. In: Hutter F, Kotthoff L, Vanschoren J, editors. Automated machine learning. Cham: Springer; 2019. p. 3–33. doi:10.1007/978-3-030-05318-5_1.
7. Yang L, Shami A. On hyperparameter optimization of machine learning algorithms: theory and practice. Neurocomputing. 2020;415:295–316. doi:10.1016/j.neucom.2020.07.061.
8. Neutatz F, Lindauer M, Abedjan Z. AutoML in heavily constrained applications. VLDB J. 2024;33:957–979. doi:10.1007/s00778-023-00820-1.
9. Bischl B, Binder M, Lang M, Pielok T, Richter J, Coors S, et al. Hyperparameter optimization: foundations, algorithms, best practices, and open challenges. WIREs Data Min Knowl Discov. 2023;13:e1484. doi:10.1002/widm.1484.
10. Franceschi L, Donini M, Perrone V, Klein A, Archambeau C, Seeger M, et al. Hyperparameter optimization in machine learning. Found Trends Mach Learn. 2025;18(6):1054–1201. doi:10.1561/2200000088.
11. Turner R, Eriksson D, McCourt M, Kiili J, Laaksonen E, Xu Z, et al. Bayesian optimization is

superior to random search for machine learning hyperparameter tuning: analysis of the black-box optimization challenge 2020 [preprint]. 2021. arXiv:2104.10201. doi:10.48550/arXiv.2104.10201.
12. Violos J, Pagoulatou T, Tsanakas S, Tserpes K, Varvarigou T. Predicting resource usage in edge computing infrastructures with CNN and a hybrid Bayesian particle swarm hyper-parameter optimization model. In: Arai K, editor. Intelligent computing. Lecture Notes in Networks and Systems, vol 284. Cham: Springer; 2021. p. 523–534. doi:10.1007/978-3-030-80126-7_40.
13. Klein A, Falkner S, Bartels S, Hennig P, Hutter F. Fast Bayesian optimization of machine learning hyperparameters on large datasets. 20th International Conference on Artificial Intelligence and Statistics (AISTATS), Fort Lauderdale, FL, USA, 2017. p. 528–536.
14. Jaderberg M, Dalibard V, Osindero S, Czarnecki WM, Donahue J, Razavi A, et al. Population based training of neural networks [preprint]. 2017. arXiv:1711.09846. doi:10.48550/arXiv.1711.09846.
15. Smith LN. A disciplined approach to neural network hyper-parameters: part 1—learning rate, batch size, momentum, and weight decay [preprint]. 2018. arXiv:1803.09820. doi:10.48550/arXiv.1803.09820.

