# A Comparative Study of Classical and Modern Boosting Algorithms for Supervised Learning

Tony Luo, Will Kim, Sichen Li, Shang Peng, Jiaheng Guo

## 1 Introduction

Boosting algorithms represent a powerful and widely adopted paradigm within supervised machine learning, particularly for tasks involving tabular data. Their ability to sequentially combine the predictions of multiple weak learners into a strong predictive model has led to implementations across numerous applications. Historically, methods such as **AdaBoost** laid the groundwork for ensemble learning, while the **Gradient Boosting Machine (GBM)** refined the approach by framing boosting as an additive model that iteratively minimizes a differentiable loss function. These classical algorithms established the fundamental principles upon which all subsequent boosting methods are built.

In recent years, the field has seen the emergence of highly optimized implementations, most notably **XGBoost** and **LightGBM**. These modern frameworks have achieved widespread acclaim, frequently dominating data science competitions due to their exceptional speed, scalability, and predictive accuracy. They build upon the core concepts of gradient boosting while introducing significant architectural and algorithmic enhancements, such as advanced regularization, histogram-based split finding, and novel tree growth strategies.

Despite the undisputed empirical success of these modern frameworks, a direct comparative analysis that systematically dissects *why* they outperform their predecessors is often lacking. Choices of algorithms are often based on popular opinion rather than solid reasons. There remains a critical gap in quantifying the practical impact of these modern innovations—such as regularization, histogram-based optimization, and sampling strategies—under a unified experimental setup.

This study aims to address this gap by conducting a comparative analysis of classical boosting algorithms, AdaBoost and GBM, alongside their modern counterparts, XGBoost and LightGBM. Our central research objective is to empirically evaluate and quantify the performance improvements offered by recent implementations and to identify the specific design innovations that underpin these enhancements. In particular, we investigate whether modern methods deliver measurable gains in predictive performance over classical approaches, how model complexity interacts with factors such as depth, shrinkage, and regularization to influence overfitting, and how computational efficiency differs among algorithms through parallelization, histogram-based optimization, and sampling strategies. Finally, we assess scalability by comparing algorithmic behavior on larger datasets, thereby highlighting how differences in system design translate into practical performance advantages. [1]

## 2 Related Works

Classical boosting methods originated with AdaBoost, which formalizes boosting as a procedure that combines weak learners under a decision-theoretic online learning framework and established strong theoretical guarantees for ensemble learning [6]. Subsequent work by Friedman cast boosting as gradient-based function approximation, introducing Gradient Boosting Machines (GBM) as stagewise additive models that iteratively minimize a differentiable loss [7]. Stochastic gradient boosting further improved robustness and computational efficiency by incorporating subsampling at each iteration, thereby regularizing the learner and accelerating training [8].

Building on these foundations, several highly optimized implementations of gradient boosting have been proposed. XGBoost popularized a scalable tree boosting system with sparsity-aware split finding, a weighted quantile sketch for approximate histograms, and explicit regularization on tree complexity [4]. LightGBM further accelerates training via gradient-based one-side sampling (GOSS) and exclusive feature bundling (EFB), enabling substantial speedups on high-dimensional and large-scale tabular datasets [11]. In parallel, CatBoost introduced ordered boosting and specialized handling of categorical features to reduce prediction shift and target leakage, often improving performance on mixed-type tabular data [12].

Several empirical studies have benchmarked these algorithms. Bentéjac et al. provide a large-scale comparison of gradient boosting variants, including XGBoost, LightGBM, CatBoost, and classical GBM, focusing on accuracy, training time, and hyperparameter sensitivity across many datasets [1]. Florek and Zagdański benchmark state-of-the-art gradient boosting implementations for classification, emphasizing tuning strategies and runtime trade-offs [5]. More broadly, Caruana and Niculescu-Mizil conduct a large empirical comparison of supervised learning algorithms, highlighting the importance of careful evaluation and hyperparameter tuning when comparing models [3].

Hyperparameter tuning itself has been explored in boosting contexts, with Bayesian optimization and TPE shown to efficiently navigate large discrete/continuous spaces for tree ensembles [13, 2]. Recent comparisons of tuning strategies report that modest trial budgets can still yield sizable gains for boosting models on tabular data [14]. There is also a body of work on boosting for imbalanced datasets that recommends careful thresholding, stratified splits, and class weights rather than aggressive resampling [10], which aligns with our conservative handling of Credit Card Fraud. On the regression side, benchmarks of gradient boosting on housing, quality-control, and materials datasets [15] note similar efficiency-accuracy trade-offs to those reported for classification.

In parallel with these studies, our work offers a narrower comparison of classical GBM with modern implementations such as XGBoost and LightGBM under a single experimental protocol. We examine how a few practical design

[1]Our code is available at: `https://github.com/jiaheng-guo/stor565`. For any questions regarding the code, please contact Jiaheng Guo at `jiaheng@unc.edu`.

choices relate to accuracy, efficiency, and overfitting on a modest set of datasets, but does not claim a comprehensive disentanglement of all boosting design factors.

## 3 Methodology

We make use of the gradient boosting algorithm, as well as the LightGBM and XGBoost algorithms which build on traditional gradient boosting. First we briefly introduce the gradient boosting algorithm following the presentation in Hastie et al. [9] and then discuss the LightGBM and XGBoost algorithms, primarily highlighting how they differ from traditional gradient boosting.

The gradient boosting model is an additive model of trees trained in a forward stagewise manner. Let the training set consist of $N$ instances, denoted $\{x_1, \dots, x_N\}$, with $x_i \in \mathbb{R}^p$ where $p$ is the number of features. We denote a tree as

$$T(x; \Theta) = \sum_{j=1}^{J} \gamma_j I(x \in R_j)$$

with parameters $\Theta = \{R_j, \gamma_j\}_1^J$, where the $R_j$ are regions that partition the space of all joint predictor variable values so that each $x_i \in R_j$ is assigned by the tree to the $j$-th leaf node, $\gamma_j$ is the output of the tree at the $j$-th leaf node, and $J$ is the number of leaves. Hence the gradient boosting model is a sum of $M$ trees

$$f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m)$$

built iteratively, where for each iteration $m$ from $m = 1$ to $m = M$ we solve

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m))$$

for a choice of loss function $L$. For general loss criteria we must approximate $L$ when optimizing in order to arrive at a simple and fast algorithm. We do so by fitting $T(x; \Theta_m)$ to the negative gradient values of $L(\mathbf{f}) = \sum_{i=1}^{N} L(y_i, f(x_i))$, where the gradient is taken with regard to

$$\mathbf{f} = \begin{bmatrix} f(x_1) & f(x_2) & \cdots & f(x_N) \end{bmatrix}^T$$

and evaluated at $\mathbf{f} = \mathbf{f}_{m-1}$, in analogy to steepest descent (whereby we would find the ideal $\mathbf{f}$ by subtracting a multiple of the negative gradient from $\mathbf{f}$ and iterating with the new value).

Recall that we find approximate solutions for the optimal $R_j$ by greedy tree induction. In other words, we recursively partition the space of joint predictor variable values by splitting on a feature $j$ at a point $s$ and considering

$$R_L(j, s) = \{x_i | x_{ij} \le s\} \text{ and } R_R(j, s) = \{x_i | x_{ij} > s\}$$

where the $R_L(j, s)$ and $R_R(j, s)$ are subsets of the points $O$ at the current node. We choose each split by finding the feature $j$ and split point $s$ that minimizes the squared error loss against the negative gradient with the optimal $\gamma_L$ and $\gamma_R$ for each region:

$$\min_{j,s}[\min_{\gamma_L} \sum_{x_i \in R_L(j,s)} (-g_i - \gamma_L)^2 + \min_{\gamma_R} \sum_{x_i \in R_R(j,s)} (-g_i - \gamma_R)^2]$$

where $g_i = \partial_{\mathbf{f}_i} L(y_i, \mathbf{f}_i)|_{\mathbf{f}_i = f_{m-1}(x_i)}$. Note that the optimal $\gamma$ for each inner minimization is given by the mean of the $-g_i$ for $x_i \in R(j, s)$:

$$\arg\min_{\gamma} \sum_{x_i \in R(j,s)} (-g_i - \gamma)^2 = \frac{1}{|R(j, s)|} \sum_{x_i \in R(j,s)} -g_i.$$

Minimizing the above objective in $j, s$ is equivalent to maximizing the variance gain at $O$, defined as

$$V_O(j, s) = \frac{1}{|O|} \left( \frac{(\sum_{x_i \in R_L(j,s)} -g_i)^2}{|R_L(j, s)|} + \frac{(\sum_{x_i \in R_R(j,s)} -g_i)^2}{|R_R(j, s)|} \right).$$

Hence, we grow the tree $T(x; \Theta_m)$ by choosing the split which maximizes the variance gain.

LightGBM alters the traditional gradient boosting method by considering only the data instances with the largest gradient values, along with a random sample of the remaining data instances when splitting (*Gradient-based One-Side Sampling*, or GOSS). This increases the efficiency of the method while still accurately estimating the variance gain.

Concretely, we rank the data instances $x_i$ at $O$ in order of greatest absolute gradient values $|g_i|$ to least, and keep the top $a \times 100\%$ data instances—which we assign to the instance subset $A$—and take a random sample of size $b \times |A^c|$ from the remaining instances—which we assign to the instance subset $B$. The variance gain at $O$ is then given by

$$\widetilde{V}_O(j, s) = \frac{1}{|O|} \left( \frac{\left(\sum_{x_i \in A_L} -g_i + \frac{1-a}{b} \sum_{x_i \in B_L} -g_i\right)^2}{|R_L(j, s)|} + \frac{\left(\sum_{x_i \in A_R} -g_i + \frac{1-a}{b} \sum_{x_i \in B_R} -g_i\right)^2}{|R_R(j, s)|} \right).$$

where $A_L = R_L(j, s) \cap A$, $A_R = R_R(j, s) \cap A$, $B_L = R_L(j, s) \cap B$, and $B_R = R_R(j, s) \cap B$.

On the other hand, XGBoost alters the traditional gradient boosting method by using a second-order approximation of $L$ in the minimization process. Ignoring regularization for simplicity, we have

$$\sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)) \approx \sum_{i=1}^{N} \left[ L(y_i, f_{m-1}(x_i)) + g_i T(x_i; \Theta_m) + \frac{1}{2} h_i T(x_i; \Theta_m)^2 \right]$$

where $h_i = \partial^2_{\mathbf{f}_i} L(y_i, \mathbf{f}_i)|_{\mathbf{f}_i = f_{m-1}(x_i)}$. Minimizing the above in $\Theta_m$ is equivalent to minimizing the objective

$$\sum_{i=1}^{N} \left[ g_i T(x_i; \Theta_m) + \frac{1}{2} h_i T(x_i; \Theta_m)^2 \right] = \sum_{j=1}^{J} \left[ \gamma_j \sum_{x_i \in R_j} g_i + \frac{1}{2} \gamma_j^2 \sum_{x_i \in R_j} h_i \right]$$

as we have only removed the constant terms. For a fixed set of regions $R_j$, the optimal weight $\hat{\gamma}_j$ is given by $\hat{\gamma}_j = -\frac{\sum_{x_i \in R_j} g_i}{\sum_{x_i \in R_j} h_i}$ which gives the optimal objective value

$$-\frac{1}{2} \sum_{j=1}^{J} \frac{\left(\sum_{x_i \in R_j} g_i\right)^2}{\sum_{x_i \in R_j} h_i}.$$

In practice, it is impossible to enumerate all possible regions $R_j$. Instead we find approximate solutions using greedy tree induction. The previous expression can be used as an impurity score to determine where to split. The information gain at $O$ is then given by

$$\frac{1}{2} \left[ \frac{\left(\sum_{x_i \in R_L} g_i\right)^2}{\sum_{x_i \in R_L} h_i} + \frac{\left(\sum_{x_i \in R_R} g_i\right)^2}{\sum_{x_i \in R_R} h_i} - \frac{\left(\sum_{x_i \in O} g_i\right)^2}{\sum_{x_i \in O} h_i} \right]$$

and we choose the split which maximizes the information gain.

LightGBM and XGBoost also both make use of methods that improve computational efficiency by reducing the number of splits which are examined when determining the optimal feature and split point. LightGBM uses a histogram-based algorithm, which buckets feature values into bins and builds a histogram during the training process. Instead of considering every possible split point, we consider splits between bins. XGBoost originally used a different method of achieving this goal, weighted quantile sketching, but by default currently also uses a histogram-based algorithm.

Additionally, we make use of the AdaBoost algorithm. The AdaBoost model is also an additive tree model. In contrast to gradient boosting models, AdaBoost reweights training samples between iterations so that misclassified samples are given more weight. AdaBoost can also be understood as forward stagewise modeling which minimizes the exponential loss $L(y, f(x)) = \exp(-y f(x))$.

## 4 Experimental Results

This section describes the experimental setup used to compare AdaBoost, GBM, XGBoost, and LightGBM. All models were evaluated under a unified and reproducible framework with consistent preprocessing, hyperparameter tuning with Optuna, and cross-validation procedures. Comparative analyses are then carried out, focusing on predictive performance and computational efficiency.

### 4.1 Datasets and Preprocessing

We evaluate the four boosting algorithms on several publicly available datasets from *Kaggle* and *UCI Machine Learning Repository* spanning 9 classification tasks and 4 regression tasks across various domains. Table 1 and 2 summarize the characteristics of the datasets we used.

| dataset | #samples | #features | #classes | domain | type | link |
|---|---|---|---|---|---|---|
| Adult Income | 48,842 | 14 | 2 | Social Science | Mixed | UCI |
| Heart Disease | 303 | 13 | 2 | Health | Mixed | UCI |
| Mushrooms | 8,124 | 22 | 2 | Biology | Categorical | Kaggle |
| Telco Customer Churn | 7,043 | 33 | 2 | Business | Mixed | Kaggle |
| Breast Cancer | 569 | 30 | 2 | Health | Numerical | UCI |
| Credit Card Fraud | 284,807 | 30 | 2 | Business | Numerical | Kaggle |
| IMDB Movie Reviews | 49,582 | $\approx$ 50,000 | 2 | Entertainment | NLP | Kaggle |
| MNIST | 70,000 | 784[2] | 10 | Computer Science | Image | UCI |
| HIGGS | 11,000,000 | 28 | 2 | Physics | Numerical | Kaggle |

Table 1: Characteristics of the Classification Datasets Used

| dataset | #samples | #features | domain | link |
|---|---|---|---|---|
| California Housing | 20,640 | 8 | Real Estate | Kaggle |
| Ames House Prices | 1,460 | 80 | Real Estate | Kaggle |
| Wine Quality | 4,898 | 11 | Business | UCI |
| Superconductivity | 21,263 | 81 | Physics | UCI |

Table 2: Characteristics of the Regression Datasets Used

These classification and regression datasets vary in sample size, feature size, and subject domain, providing different scenarios for comprehensive evaluation of the boosting algorithms. The data types involved include numerical, categorical, images, and text. Specifically, we included two datasets, IMDB Movie Reviews, which consists of text data of audience attitudes towards movies, and MNIST, the classical handwritten digits dataset, to examine the versatility of these algorithms on NLP and CV classification tasks. Moreover, the HIGGS dataset contains 11 million

[2]We used PCA to reduce the 784 dimensions in our experiment.

samples of particle collision data, making it a significant dataset for evaluating the performance of boosting algorithms on large-scale numerical data. The Credit Card Fraud dataset is highly imbalanced, with only 492 frauds out of 284,807 transactions. All datasets were first processed through the following shared pipeline and were subsequently adjusted as needed for each dataset:

(1) Missing numerical features were imputed using medians; missing categorical features were imputed using the most frequent category.

(2) Categorical variables were one-hot encoded for AdaBoost, GBM, and XGBoost, while LightGBM additionally took advantage of its built-in categorical support.

(3) Numerical features were left unscaled, consistent with tree-based model characteristics.

(4) Training and testing splits were generated using fixed random seeds with an 80/20 ratio.

We further processed the NLP and CV datasets. For IMDB Movie Reviews, we lowercased text, stripped punctuation, removed English stopwords, and applied TF-IDF with sublinear term frequency, `max_df=0.95`, `min_df=5`, and a capped 50k vocabulary; the output is a sparse matrix consumed directly by tree models (we keep `with_mean=False` in any scaler to respect sparsity). For MNIST, we normalized pixel values to [0,1], flattened 28×28 images to 784-dimensional vectors, and then applied PCA (`whiten=False`) keeping 95% variance, reducing dimensionality prior to fitting. Within the 5-fold CV used for evaluation and Optuna tuning, preprocessing steps are refit per fold to prevent target leakage. Class imbalance (e.g., Credit Card Fraud) is left unchanged except for stratified splitting so that metrics reflect performance on the natural distribution; we do not apply class weights or resampling in the main experiments. These choices mirror the code in `src/preprocess.py` and the pipelines assembled in `src/modeling.py`, ensuring consistency between reporting and implementation.

## 4.2 Hyperparameter Tuning and Evaluation Metrics

Hyperparameter optimization was carried out with *Optuna* (TPE sampler). For each dataset–algorithm pair we first ran a baseline model with fixed, conservative settings, then launched an Optuna study (5-fold cross-validation) that maximized ROC–AUC for classification or minimized RMSE for regression. The search varied the learning rate, number of estimators, tree depth/leaf parameters, subsampling, column subsampling, and minimum child size (or leaf/min_data in LightGBM) to mirror the knobs exposed by each library. We kept a single 5-fold CV loop (no nesting) and used fixed seeds for reproducibility.

For classification, we report accuracy, F1-score, and ROC–AUC (one-vs-rest for multiclass); for regression, RMSE, MAE, and $R^2$. Training time per fold is recorded and later visualized to show efficiency differences. We did not attempt to report inference latency or feature-importance stability beyond what the libraries provide by default.

The Optuna search spaces we defined are: learning rate in $[10^{-3}, 0.3]$, estimators in $[50, 400]$, max depth in $\{2, \ldots, 10\}$ (or leaf-wise parameters for LightGBM), subsample and colsample in $[0.6, 1.0]$, and min child weight (or min_data_in_leaf) in $[1, 50]$. Each study used 20 trials with a 30-minute cap per dataset–algorithm pair; pruning was enabled to stop underperforming configurations early. For multiclass MNIST, we used one-vs-rest ROC–AUC in the objective to ensure probability calibration across classes. All folds shared deterministic seeds and stratified splits where applicable to maintain consistent class proportions.

## 4.3 Comparative Analysis on Classification Tasks

In this subsection, we present the comparative results of AdaBoost, GBM, XGBoost, and LightGBM on the classification datasets described in Section 4. The AUC scores across different datasets and (untuned) algorithms are summarized in Figure 1.

From the results, we observe that modern boosting algorithms generally outperform their classical counterparts across most datasets, while there is no single algorithm that consistently dominates all others. Notably, XGBoost and LightGBM exhibit significant improvements in AUC scores on larger datasets such as HIGGS and Credit Card Fraud, likely due to their optimized handling of large-scale data through parallelization and histogram-based split finding. On smaller datasets like Heart Disease and Breast Cancer, the performance gap narrows, suggesting that the advantages of modern algorithms may be more pronounced in high-dimensional or large-sample scenarios. Results in terms of the other metrics (accuracy and F1-score) follow similar trends; due to space limitations, we only present AUC results here.

We also examine how performance is affected by hyperparameter choices, particularly learning rate and tree depth. Figure 2 illustrates the AUC results for algorithms tuned with 20 Optuna trials across all nine classification datasets. Figure 3 shows the AUC gain from hyperparameter tuning for each algorithm. We observe that all four algorithms benefit from hyperparameter optimization, with GBM and LightGBM showing the most significant improvements on several datasets. However, due to the limitation of computational resources, we only performed 20 trials for each dataset-algorithm pair, which may not be sufficient to fully explore the hyperparameter space and converge to the global optimum. Future work could involve more extensive tuning to better understand the sensitivity of each algorithm to hyperparameter settings.

## 4.4 Comparative Analysis on Regression Tasks

We evaluate AdaBoost, GBM, XGBoost, and LightGBM on the four regression datasets: California Housing, Ames, Wine Quality, and Superconductivity. Figure 4 reports baseline RMSEs without tuning. LightGBM and XGBoost lead on all datasets, with larger margins on the higher-dimensional Ames and Superconductivity tasks, while AdaBoost and GBM trail but remain competitive on Wine Quality and California Housing.

After 20 Optuna trials per dataset–algorithm pair, all methods improve (Figure 5). The biggest drops come from GBM and AdaBoost, which close part of the gap to the histogram-based libraries on the smaller tabular datasets. MAE

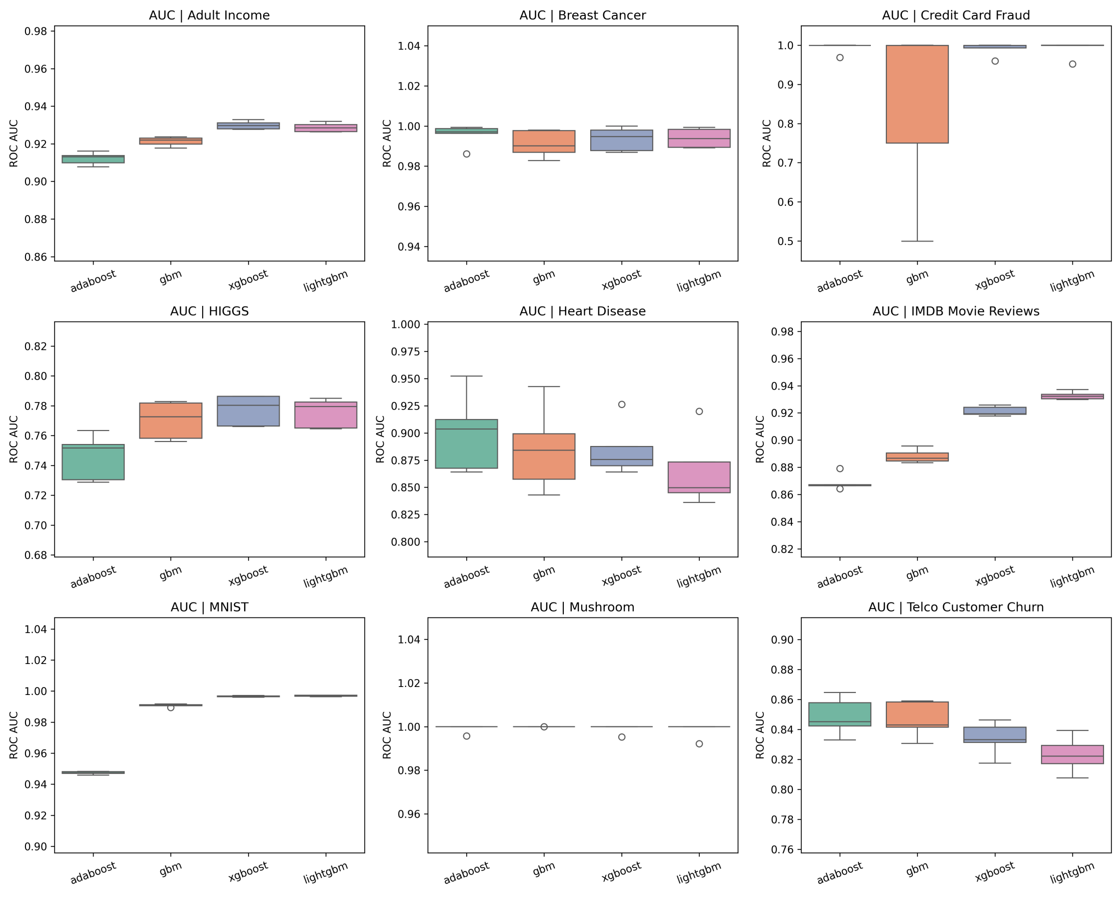

Figure 1: AUC Comparison Across Baseline Boosting Algorithms

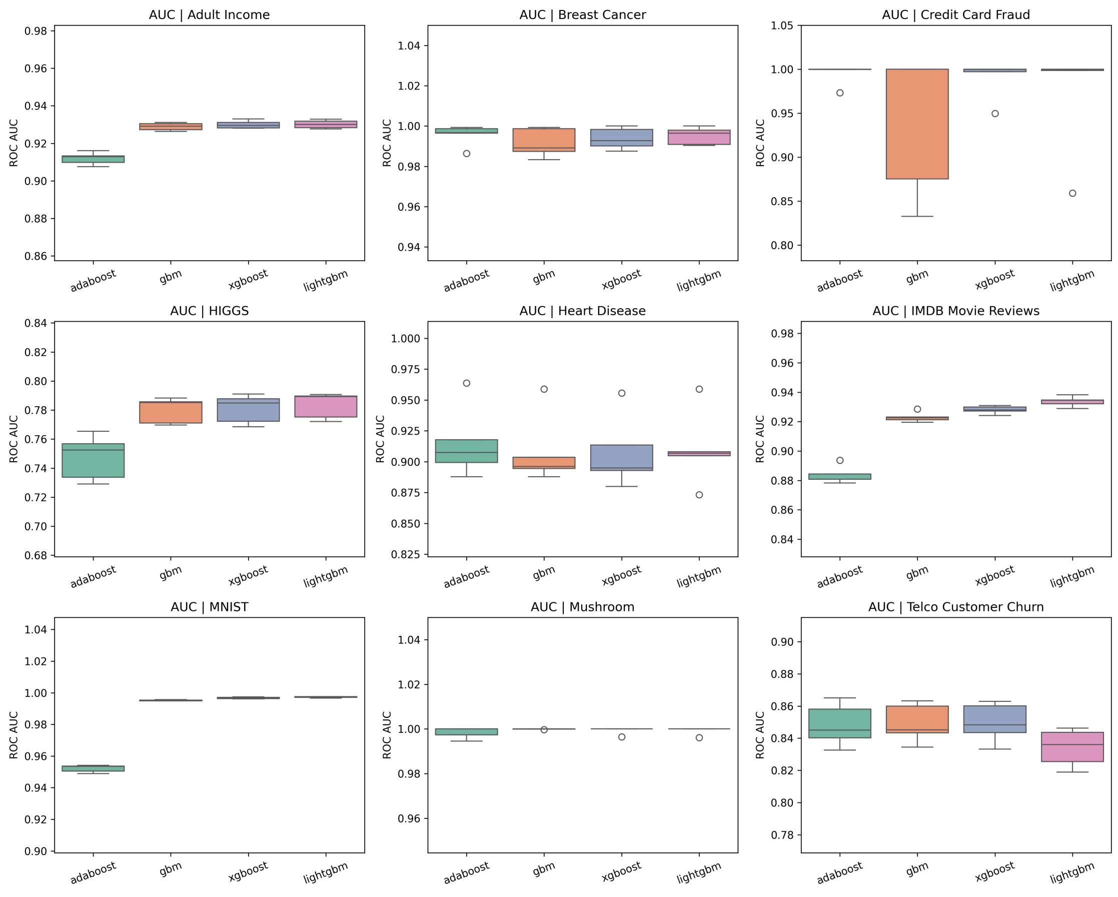

Figure 2: AUC Comparison Across Optimized Boosting Algorithms (20 Optuna Trials)

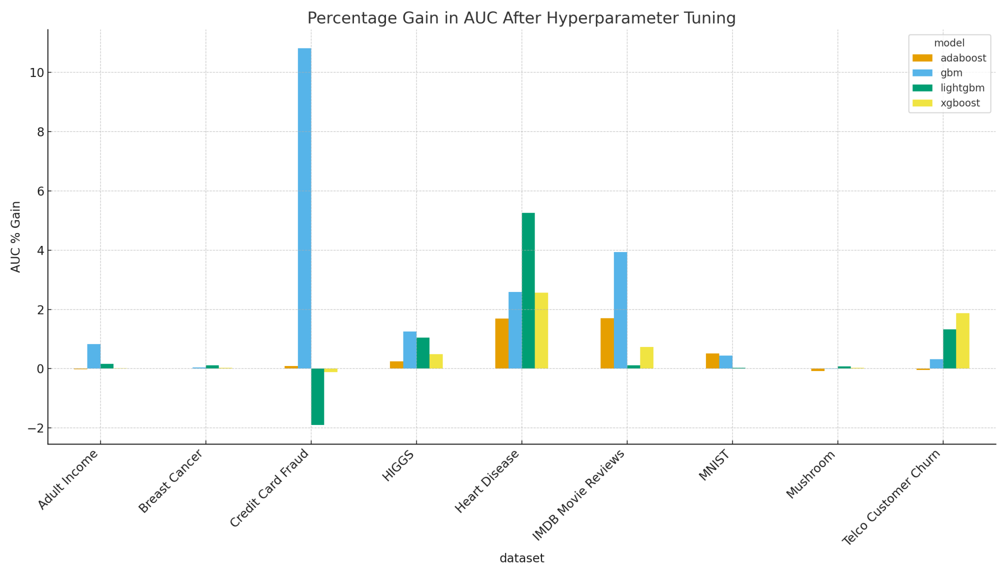

Figure 3: AUC Gain from Hyperparameter Tuning Across Boosting Algorithms

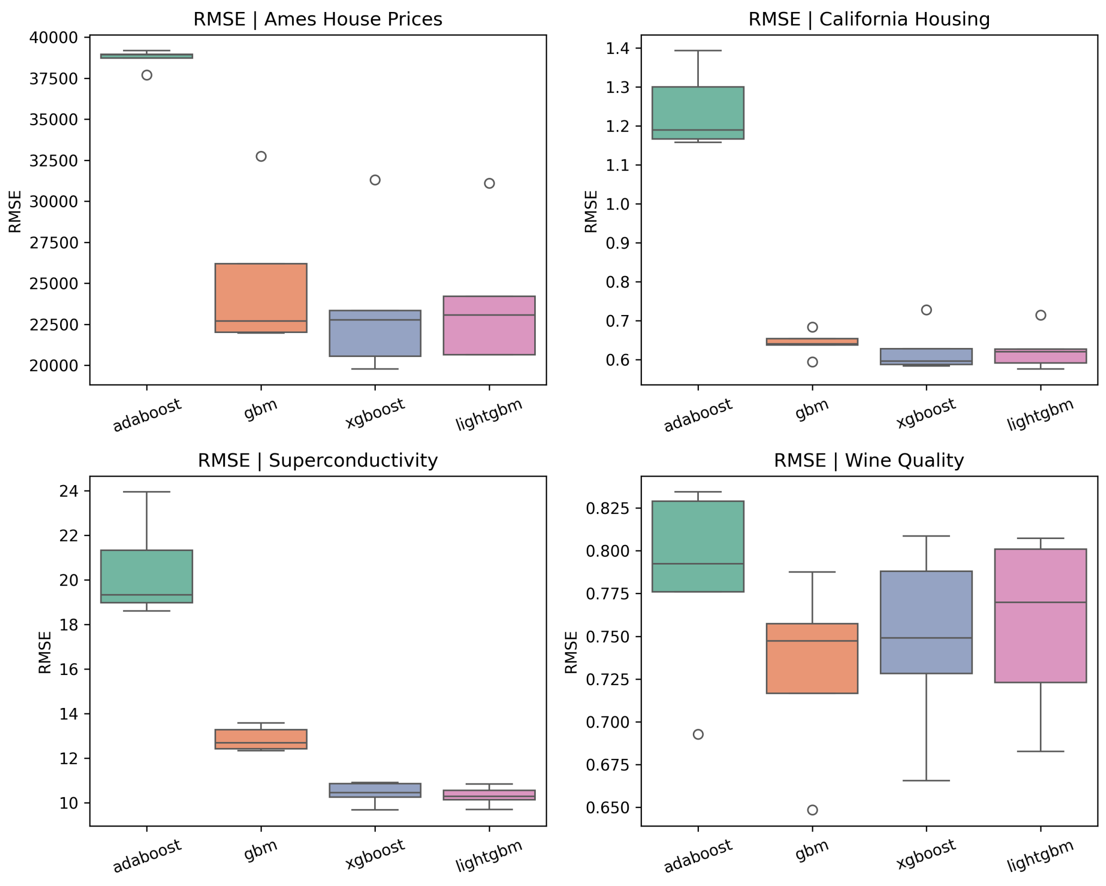

Figure 4: Baseline RMSE Across Boosting Algorithms on Regression Datasets

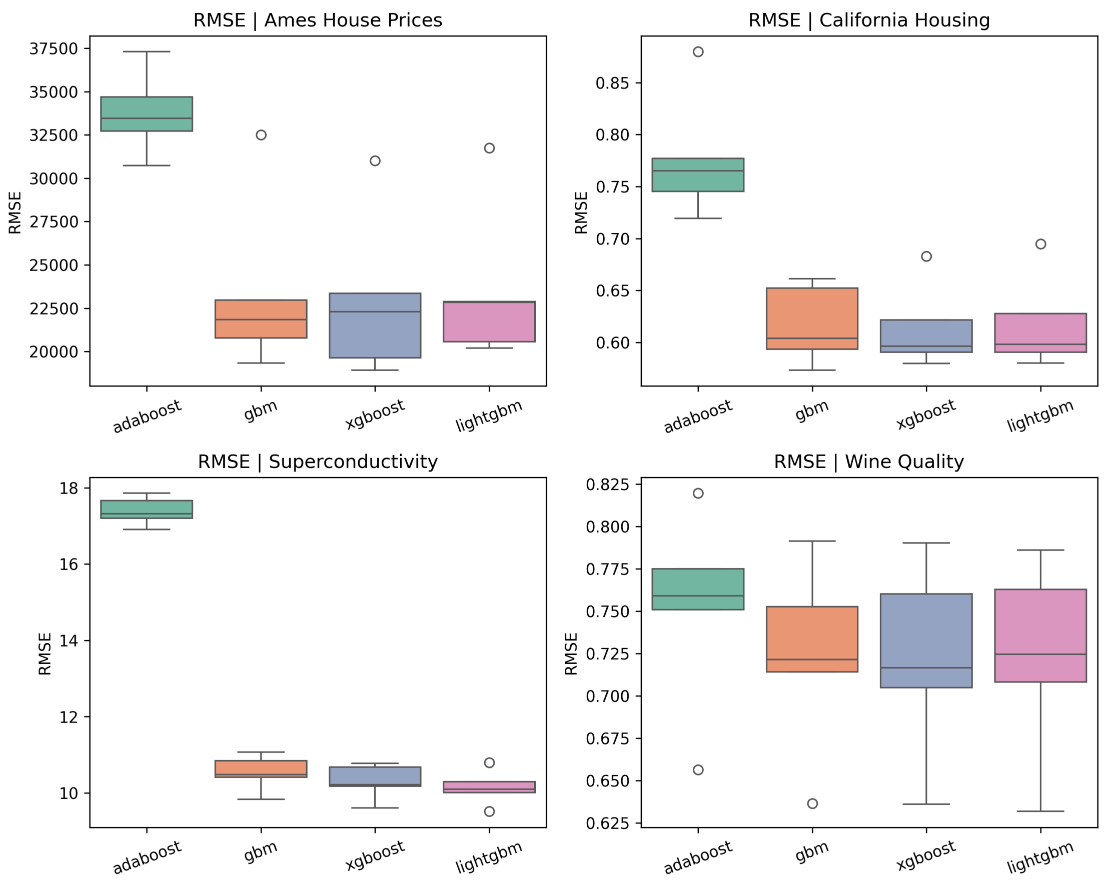

Figure 5: Optuna-Tuned RMSE Across Boosting Algorithms on Regression Datasets

and $R^2$ (not shown) mirror the same ordering, suggesting the tuning benefits are consistent across metrics. Overall, tuned LightGBM attains the lowest errors on three of four datasets, and XGBoost stays competitive on Wine Quality, echoing the classification patterns.

To complement RMSE, we examined MAE and $R^2$ boxplots (omitted for brevity). MAE emphasized the robustness gains from tuning on Wine Quality and California Housing, while $R^2$ highlighted the larger variance explained by LightGBM on Ames and Superconductivity after tuning. We observed that the variance across CV folds shrinks notably for the tuned GBM/AdaBoost runs, indicating better performance stability across CV folds once learning rate, tree depth, and subsampling are calibrated. Taken together, the regression results reinforce the efficiency advantages of histogram-based methods, but also show that classical GBM can recover competitive accuracy with modest, targeted hyperparameter searches.

## 4.5 Comparative Analysis on Computational Efficiency

In terms of computational efficiency, we have selected several representative plots for training time. We observed that modern algorithms, XGBoost and LightGBM, consistently demonstrate faster training times across datasets, attributable to their histogram-based approach and leaf-wise tree growth strategy. The classical algorithms, though demonstrated comparative training efficiency on several smaller datasets, tend to lag behind as data size increases. The training time comparison is illustrated in Figure 6.

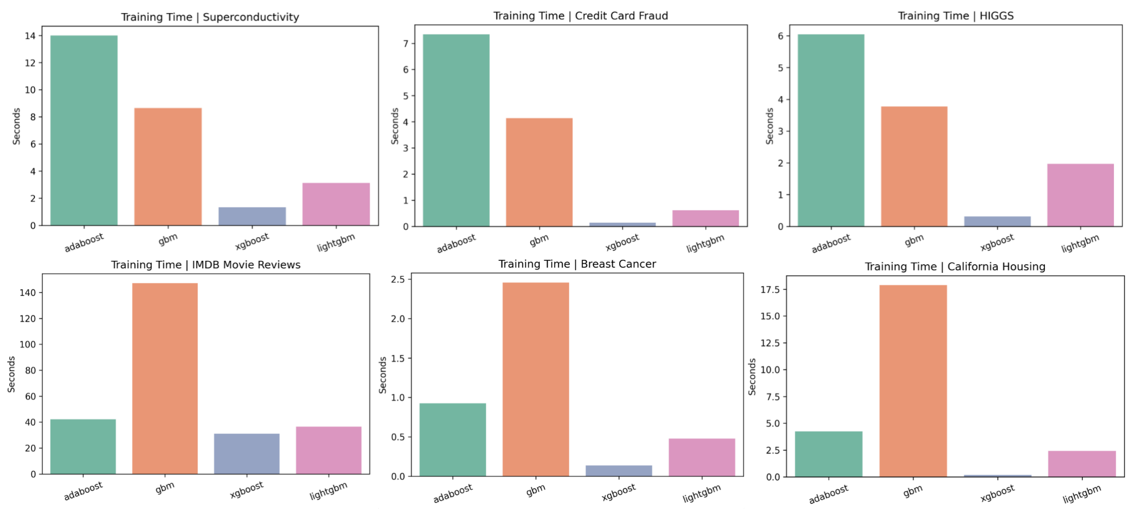

Figure 6: Selected Training Time Comparison

### 4.6 Additional Experimental Results

In the `static` folder of our GitHub repository, we saved additional experimental results that were not included in the main text due to space limitations, including accuracy and F1-score for classification tasks, as well as MAE and $R^2$ for regression tasks, all provided as figures and CSV files. Please refer to the folder for the complete results.

## 5 Conclusion and Future Direction

Based on the comparison of classical and modern boosting algorithms, we systematically evaluated the impact of recent design innovations. Our key findings confirm that modern frameworks represent a significant advancement in both performance and efficiency. Across a suite of classification and regression tasks, XGBoost and LightGBM consistently achieved superior predictive accuracy compared to their predecessors. Specifically, histogram-based methods provided dramatic speedups in computational efficiency, while the built-in L1 and L2 regularization in XGBoost offered more robust control over model complexity and overfitting. Feature-importance stability was not exhaustively assessed; we focus instead on predictive metrics and runtime.

Aside from these clear results, there do exist several limitations. First, the analysis was confined to a selection of publicly available Kaggle datasets. While diverse, they may not capture the full spectrum of data characteristics in all real-world applications, such as datasets with extremely high-cardinality categorical features or non-standard data distributions. Second, while we employed a standardized hyperparameter tuning process, the vastness of the parameter search space means we cannot guarantee that the absolute optimal configuration for each algorithm was discovered. It is possible that classical models, in particular, could see further improvement with more advanced tuning.

Based on these findings and limitations, several future directions arise. A possible next step is to extend this comparative analysis to a wider and more diverse range of datasets, including those from different domains such as finance or bioinformatics, to further validate the scalability and generalizability of these algorithms. Potential techniques for hyperparameter tuning, such as Bayesian optimization, could be used to discover the peak performance of each model. Another valuable extension is to benchmark these boosting methods against other competing model classes, particularly deep learning architectures designed for tabular data, to better situate their performance within the broader machine learning landscape.

## References

[1] C. Bentéjac, A. Csörgő, and G. Martínez-Muñoz. A comparative analysis of gradient boosting algorithms. *Artificial Intelligence Review*, 54(3):1937–1967, 2021.

[2] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kégl. Algorithms for hyper-parameter optimization. In *Advances in Neural Information Processing Systems*, volume 24, pages 2546–2554, 2011.

[3] R. Caruana and A. Niculescu-Mizil. An empirical comparison of supervised learning algorithms. In *Proceedings of the 23rd International Conference on Machine Learning (ICML 2006)*, pages 161–168. ACM, 2006.

[4] T. Chen and C. Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. Association for Computing Machinery.

[5] P. Florek and A. Zagdański. Benchmarking state-of-the-art gradient boosting algorithms for classification. *arXiv preprint arXiv:2305.17094*, 2023.

[6] Y. Freund and R. E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Computer and System Sciences*, 55(1):119–139, 1997.

[7] J. H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189–1232, 2001.

[8] J. H. Friedman. Stochastic gradient boosting. *Computational Statistics & Data Analysis*, 38(4):367–378, 2002.

[9] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York, New York, NY, 2009.

[10] H. He and E. A. Garcia. Learning from imbalanced data. *IEEE Transactions on Knowledge and Data Engineering*, 21(9):1263–1284, 2009.

[11] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T.-Y. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*, pages 3146–3154, Long Beach, CA, USA, 2017. Curran Associates, Inc.

[12] L. Prokhorenkova, G. Gusev, A. Vorobev, A. V. Dorogush, and A. Gulin. Catboost: unbiased boosting with categorical features. *Advances in neural information processing systems*, 31, 2018.

[13] J. Snoek, H. Larochelle, and R. P. Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, volume 25, pages 2951–2959, 2012.

[14] F. Yang and A. Shami. On hyperparameter optimization of machine learning algorithms: Theory and practice. *IEEE Access*, 8:164137–164153, 2020.

[15] Y. Zhao, H. Liu, and W. Zhang. Benchmarking gradient boosting models for regression tasks. *arXiv preprint arXiv:2006.00000*, 2020.