Neural Routed Boosting: Robust Learning against Heteroscedastic Noise

Puspak Chakraborty[1] **Arun Rajkumar**[1]

[1]Data Science and Artificial Intelligence Dept., Indian Institute of Technology, Chennai, India

Abstract

Boosting algorithms such as AdaBoost have achieved widespread success by iteratively focusing on hard-to-classify instances. However, this aggressive re-weighting mechanism makes them susceptible to label noise and overfitting. While robust variants like Conditional Boosting handle this by estimating conditional risk, they rely on global statistical approximations. We propose Neural Routed Boosting, a novel ensemble framework that addresses heteroscedastic noise through a structural approach. Neural Routed Boosting utilizes a lightweight neural network to partition the input space into geometrically coherent regions, training specialized weak learners for each. This isolates noisy subspaces, preventing them from corrupting the decision boundaries of clean regions. We prove that the composite model of a neural router and region-specific experts constitutes a valid weak learner under the boosting framework, guaranteeing training error convergence. Experimental results on synthetic and real-world datasets demonstrate that Neural Routed Boosting outperforms traditional and robust boosting baselines, including Conditional Boosting, in high-noise environments.

1 INTRODUCTION

Boosting is a powerful ensemble technique that combines weak classifiers into a strong predictor. The seminal AdaBoost algorithm [Freund and Schapire, 1996] simultaneously reduces bias and variance. However, it suffers from a critical weakness: its exponential re-weighting of misclassified samples causes severe overfitting in the presence of label noise. Outliers and mislabeled instances are repeatedly emphasized, causing the model to diverge from the true decision boundary in an attempt to fit erroneous data [Dietterich, 2000]. As noted by Xiao et al. [2018], the prevalence of label noise is a well-documented challenge in practical machine learning. Obtaining reliable ground-truth labels is often both expensive and difficult, to the extent that in critical domains such as biomedical data, perfect training labels are considered almost impossible to obtain. To address this issue, several robust variants of boosting have emerged. Early approaches modified the underlying loss function or weighting mechanics. For example, Friedman et al. [2000] introduced LogitBoost and Gentle AdaBoost. These methods replace the aggressive exponential loss with gentler, outlier-resistant alternatives. In contrast, Smooth Boosting [Servedio, 2003] and the penalized updating scheme of Zhang et al. [2008] imposed mathematical restrictions on instance distributions to prevent over-concentration on noisy samples. Other methods attempted dynamic data filtering, where algorithms like Edited AdaBoost [Gao and Gao, 2010] and ND-AdaBoost [Cao et al., 2012] utilize k-NN heuristics to hard-delete or relabel suspicious instances during training. These hard thresholds often discard valuable boundary information.

Alternatively, Conditional Boosting (CB-AdaBoost) [Xiao et al., 2018] introduced a statistical approach. It down-weights samples based on their estimated "Conditional Risk" of label corruption. While effective, CB-AdaBoost relies on global risk estimation. This global approximation can be unstable in high-dimensional spaces or under complex, heteroscedastic noise.[1]

To address these limitations, we introduce Neural Routed Boosting (NRB). Unlike CB-AdaBoost's reliance on global statistical weighting, NRB employs a "divide and conquer" strategy by using a lightweight "Router" network to partition the input space. By isolating noisy regions into specific geometric buckets and training specialized weak learners for each, this architecture prevents high-noise regions from

[1]Heteroscedastic noise refers to a condition where the variance or distribution of label noise is non-uniform and varies as a function of the input features $\mathbf{x}$.

Proceedings of the 42nd Conference on Uncertainty in Artificial Intelligence (UAI 2026), PMLR 337:961–980.

corrupting the decision boundaries of clean regions. Our main contributions are as follows:

1. We propose Neural Routed Boosting (NRB), which uses a neural routing mechanism to handle region-specific noise.
2. We establish theoretical guarantees for NRB, proving that the joint formulation of the neural router and localized base models satisfies the weak learning condition, thereby ensuring the convergence of the training error.
3. We demonstrate that NRB's partitioning strategy offers a stronger defense against heteroscedastic noise than the statistical weighting of CB-AdaBoost.
4. We provide experimental results on synthetic and real-world datasets, showing that NRB achieves superior accuracy in high-noise regimes.

2 RELATED WORKS

Standard boosting algorithms, such as AdaBoost [Freund and Schapire, 1996], minimize the exponential loss function. While effective on clean data, the exponential loss is not robust to outliers. **Robust Boosting:** Several variations have attempted to robustify boosting. BrownBoost [Freund, 2001] uses a non-convex potential loss function that allows the algorithm to ignore examples with large cumulative margins (large errors). Vezhnevets and Vezhnevets [2005] proposed Modest AdaBoost, which alters the weak learner's objective to trade off between weighted training error and generalization capability. MadaBoost [Domingo and Watanabe, 2000] modifies the weighting scheme to limit the influence of outliers. Unlike these methods, NRB does not fundamentally alter the loss function but rather the *structure* of the weak learner, allowing for localized adaptation. **Region-based Boosting:** Partitioning the feature space for ensemble learning has roots in decision trees [Breiman et al., 1984] and local boosting variants [Zhang and Zhang, 2008], which adapt weights using localized neighborhood information. However, NRB differs by decoupling the spatial partitioning logic from the classification objective, a structure inspired by Mixture of Experts [Jacobs et al., 1991]. By utilizing a neural router, NRB enables complex, non-linear region boundaries that simple stumps cannot achieve, projecting data into geometrically cohesive spaces for the local experts.

3 PRELIMINARIES

We consider the standard binary classification setting. Let $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_N, y_N)\}$ be the training set, where $\mathbf{x}_i \in \mathcal{X} \subseteq \mathbb{R}^d$ represents the feature vector and $y_i \in \{-1, +1\}$ is the corresponding class label.

3.1 STANDARD ADABOOST AND THE NOISE PROBLEM

The core principle of boosting [Freund and Schapire, 1996] is to construct a strong classifier by combining a sequence of weak hypotheses. At each round $t \in \{1, \ldots, T\}$, the algorithm maintains a distribution D_t over the examples. A *weak learner* returns a hypothesis $h_t : \mathcal{X} \to \{-1, +1\}$. Its weighted error $\epsilon_t = \sum_{i=1}^{N} D_t(i)\mathbb{I}(h_t(\mathbf{x}_i) \neq y_i)$ must be strictly better than random guessing. That is, $\epsilon_t \leq 0.5 - \delta_t$ for some edge $\delta_t > 0$. Standard AdaBoost minimizes the exponential loss $\mathcal{L}_{\exp}(y, F(\mathbf{x})) = \exp(-yF(\mathbf{x}))$. It assigns a voting weight $\alpha_t = \frac{1}{2}\ln(\frac{1-\epsilon_t}{\epsilon_t})$ to each hypothesis based on its error. The final ensemble is the weighted sum $F(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})$. The discrete prediction is then $H(\mathbf{x}) = \text{sign}(F(\mathbf{x}))$. We detail the standard AdaBoost algorithm in Appendix I.

A fundamental property of this scheme is that the training error decreases exponentially fast.

Theorem 1 (Boosting Convergence [Freund and Schapire, 1996])**.** *The training error of the final classifier ($H(\mathbf{x})$) is upper bounded by:*

$$\frac{1}{N}\sum_{i=1}^{N}\mathbb{I}(H(\mathbf{x}_i) \neq y_i) \leq \prod_{t=1}^{T}\sqrt{1-4\delta_t^2} \leq \exp\left(-2\sum_{t=1}^{T}\delta_t^2\right). \tag{1}$$

This inequality guarantees the training error converges to zero, provided the weak learners maintain a positive edge. However, this powerful convergence becomes a liability under label noise. To satisfy the theoretical bound, AdaBoost exponentially up-weights "hard" examples. When these hard examples are mislabeled outliers, the algorithm distorts the true decision boundary to fit them. This behavior causes overfitting and poor generalization.

3.2 ROBUSTNESS VIA CONDITIONAL RISK (CB-ADABOOST)

To address the vulnerability of standard boosting to label noise, Xiao et al. [2018] introduced the **Conditional Boosting (CB-AdaBoost)** framework. They argue that the contribution of a sample should depend on its *Conditional Risk*, incorporating the uncertainty of the observed labels. Instead of blind exponential up-weighting, CB-AdaBoost modifies the optimization objective by introducing a label confidence parameter $\gamma_i = P(Z_i = y_i|\mathbf{x}_i)$, which represents the probability that the observed label y_i is correct. The algorithm replaces the standard exponential loss with the conditional risk loss

$$\mathcal{L}_\gamma(y_i, F(\mathbf{x}_i)) = \gamma_i \exp(-y_i F(\mathbf{x}_i)) + (1-\gamma_i)\exp(y_i F(\mathbf{x}_i)). \tag{2}$$

This modified loss function allows the model to make a smooth transition between full acceptance and full rejection of a sample's label. To minimize this objective, CB-AdaBoost maintains two sets of weights for each instance, w_{1i} and w_{2i}, corresponding to the assumption that the observed label is trusted versus mislabeled. The training process dynamically relabels instances based on their current weight differences and calculates a confidence-adjusted voting weight β_t that dampens the influence of noisy samples. This prevents the model from "chasing" outliers by modestly combining base classifiers. The complete procedure is outlined in Algorithm 1:

Algorithm 1: CB-AdaBoost Algorithm

Input: Training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^N$, label confidences $\{\gamma_i\}_{i=1}^N$, iterations T.

Initialization:

for *all* $i \in \{1, \ldots, N\}$ **do**

$w_{1i}^{(1)} \leftarrow \gamma_i, \quad w_{2i}^{(1)} \leftarrow 1 - \gamma_i$

end

Set initial distribution $D_i^{(1)} \leftarrow |w_{1i}^{(1)} - w_{2i}^{(1)}|/S_1$, where $S_1 = \sum_{i=1}^N |w_{1i}^{(1)} - w_{2i}^{(1)}|$

for $t = 1$ **to** T **do**

`// 1. Prepare Weights & Targets`

Relabel instances to form $S' = \{(\mathbf{x}_i, y_i')\}$, where $y_i' = \text{sign}((w_{1i}^{(t)} - w_{2i}^{(t)})y_i)$

`// 2. Train Weak Learner`

Train weak learner h_t on S' using data distribution $D^{(t)}$

`// 3. Calculate Voting Weight`

$$\beta_t \leftarrow \frac{1}{2} \ln \left(\frac{\sum_{h_t = y_i} w_{1i}^{(t)} + \sum_{h_t \neq y_i} w_{2i}^{(t)}}{\sum_{h_t \neq y_i} w_{1i}^{(t)} + \sum_{h_t = y_i} w_{2i}^{(t)}} \right)$$

if $\beta_t < 0$ **then**

break (Learner worse than random guessing)

end

`// 4. Update Internal Weights`

$w_{1i}^{(t+1)} \leftarrow w_{1i}^{(t)} \exp(-y_i \beta_t h_t(\mathbf{x}_i))$

$w_{2i}^{(t+1)} \leftarrow w_{2i}^{(t)} \exp(y_i \beta_t h_t(\mathbf{x}_i))$

`// 5. Update Data Distribution`

$D_i^{(t+1)} \propto |w_{1i}^{(t+1)} - w_{2i}^{(t+1)}|$

end

Output: $H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^T \beta_t h_t(\mathbf{x})\right)$

return $H(\mathbf{x})$

In our work, we adopt this mathematically sound confidence-based weighting strategy but apply it within a structurally routed framework to map and handle region-specific label noise.

4 METHODOLOGY: NEURAL ROUTED BOOSTING

In this section, we detail the Neural Routed Boosting (NRB) algorithm. Our framework employs an adaptive routing mechanism that partitions the feature space at each round, integrated with a robust *Confidence-Based* weighting strategy inspired by the Conditional Boosting (CB-AdaBoost) framework [Xiao et al., 2018].

4.1 THE NRB ARCHITECTURE

The core innovation of NRB is the decomposition of the weak learner h_t into a composite function of a *Router* and a set of *Experts*. **The Router ($\mathcal{R}$):** We employ a neural network $\mathcal{R}(\mathbf{x}; \Theta)$ that maps an input $\mathbf{x}$ to a bucket index $k \in \{1, \ldots, K\}$. The router divides the input space into K disjoint regions. **The Experts ($g_{t,k}$):** For each round t and bucket k, a specialized weak learner $g_{t,k}(\mathbf{x})$ is trained on the data falling into that region during that round.

4.2 ALGORITHM DESCRIPTION

The detailed training procedure is presented in Algorithm 2. Following initialization of router weights and k-NN confidence estimation, each boosting round integrates geometric partitioning and ensemble learning through the following six steps:

1. Prepare Weights & Targets: Following CB-AdaBoost, we compute the effective target label y_i^* and normalized data distribution D_t using the absolute difference between the trusted weight w_1 and mislabeled weight w_2.

2. Route Samples: The neural router assigns each instance $\mathbf{x}_i$ to a bucket via the $\arg\max$ of its output probabilities. This splits the data into K disjoint geometric partitions ($\mathcal{D}_{k,t}$) for the t-th round.

3. Train Bucket Experts: A localized weak learner $h_{t,k}$ is trained on partition $\mathcal{D}_{k,t}$ using the current weight distribution (empty buckets receive a null learner). The global composite learner h_t then pairs the router's assignments with these local experts.

4. & 5. Compute Coefficient & Update Weights: We calculate the confidence-adjusted voting weight α_t and exponentially update the dual weights w_1 and w_2 based on the composite learner's predictions.

6. Update Neural Router: To provide a feedback loop, we evaluate the K experts' predictive correctness to form a target softmax distribution Q. The router's parameters Θ are updated via gradient descent to minimize the cross-entropy loss $\mathcal{L}_{CE}$, teaching the network to route samples toward the most capable experts.

Algorithm 2: Neural Routed Boosting

Input: Training data $\mathcal{D} = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$, Rounds T, Buckets K, Neighbors k_{nn}

Initialization:

Initialize Router $\mathcal{R}(\mathbf{x}; \Theta)$ with random weights.

Estimate confidence γ_i for each $\mathbf{x}_i$ using k_{nn}-Nearest Neighbors.

Initialize dual weights: $w_{1,i}^{(1)} \leftarrow \gamma_i, \quad w_{2,i}^{(1)} \leftarrow 1 - \gamma_i$.

for $t = 1$ **to** T **do**

`// 1. Prepare Weights & Targets (CB-AdaBoost)`

$y_i^* \leftarrow \text{sign}(w_{1,i}^{(t)} - w_{2,i}^{(t)}) \cdot y_i$

$D_t(i) \propto |w_{1,i}^{(t)} - w_{2,i}^{(t)}|$, normalized so $\sum D_t(i) = 1$

`// 2. Route Samples (Dynamic Step)`

Store state $\mathcal{R}^{(t)} \leftarrow \mathcal{R}(\cdot; \Theta)$ and assign $k_i = \arg\max_k \mathcal{R}_k(\mathbf{x}_i; \Theta)$

Define disjoint partitions: $\mathcal{D}_{k,t} = \{i \mid k_i = k\}$ for $k = 1, \ldots, K$

`// 3. Train Bucket Experts`

for $k = 1$ **to** K **do**

if $\mathcal{D}_{k,t} \neq \emptyset$ **then**

$h_{t,k} \leftarrow \arg\min_{h \in \mathcal{H}} \sum_{i \in \mathcal{D}_{k,t}} D_t(i) \mathbb{I}(h(\mathbf{x}_i) \neq y_i^*)$

end

else

$h_{t,k} \leftarrow$ Null learner (random guessing)

end

end

Define $\kappa(\mathbf{x}) = \arg\max_k \mathcal{R}_k(\mathbf{x}; \Theta)$ and $h_t(\mathbf{x}) = h_{t,\kappa(\mathbf{x})}(\mathbf{x})$

`// 4. Compute Coefficient (CB-AdaBoost)`

$\alpha_t = \frac{1}{2} \ln \left(\frac{\sum_{i: h_t(\mathbf{x}_i) = y_i^*} w_{1,i}^{(t)} + \sum_{i: h_t(\mathbf{x}_i) \neq y_i^*} w_{2,i}^{(t)}}{\sum_{i: h_t(\mathbf{x}_i) \neq y_i^*} w_{1,i}^{(t)} + \sum_{i: h_t(\mathbf{x}_i) = y_i^*} w_{2,i}^{(t)}} \right)$

if $\alpha_t < 0$ **then**

break (Learner worse than random guessing)

end

`// 5. Update Weights`

$w_{1,i}^{(t+1)} \leftarrow w_{1,i}^{(t)} \exp(-\alpha_t h_t(\mathbf{x}_i) y_i)$

$w_{2,i}^{(t+1)} \leftarrow w_{2,i}^{(t)} \exp(+\alpha_t h_t(\mathbf{x}_i) y_i)$

Normalize $w_1^{(t+1)}, w_2^{(t+1)}$

`// 6. Update Neural Router`

Calculate correctness: $s_{i,k} = P(y = y_i^* \mid \mathbf{x}_i; h_{t,k})$

Target distribution: $Q_{i,k} = \frac{\exp(s_{i,k})}{\sum_{j=1}^K \exp(s_{i,j})}$

Update Θ via Gradient Descent to minimize Cross-Entropy $\mathcal{L}_{CE}(Q, \mathcal{R}(\mathbf{x}; \Theta))$

end

Final Prediction: $F(\mathbf{x}_{new}) = \sum_{t=1}^T \alpha_t h_{t,k_t}(\mathbf{x}_{new})$ where $k_t = \arg\max_k \mathcal{R}_k^{(t)}(\mathbf{x}_{new})$

return $\text{sign}(F(\mathbf{x}_{new}))$

4.3 THEORETICAL ANALYSIS

The convergence of NRB relies on the fact that the composite model $h_t(\mathbf{x})$ satisfies the weak learner assumption.

Theorem 2 (Composite Weak Learner)**.** *Let the input space $\mathcal{X}$ be partitioned into K disjoint regions $S_1, \ldots, S_K$ by the router $\mathcal{R}$. Let $W_k = \sum_{i \in S_k} D_t(i)$ be the total probability mass of region k. If each local expert $h_{t,k}$ satisfies the weak learning assumption (accuracy > 0.5) with respect to the local distribution $D_t|_{S_k}$, then the composite learner h_t satisfies the weak learning assumption on the global distribution D_t.*

Proof. Let $\epsilon_{t,k}$ denote the weighted error of the k-th expert on its local partition. By definition, the local error is normalized by the region's weight

$$\epsilon_{t,k} = \frac{1}{W_k} \sum_{i \in S_k} D_t(i) \mathbb{I}(h_{t,k}(\mathbf{x}_i) \neq y_i). \tag{3}$$

The assumption that each expert is a weak learner implies that for all k where $W_k > 0$, we have $\epsilon_{t,k} < 0.5$.

The global error ϵ_t of the composite learner $h_t(\mathbf{x})$ is the sum of errors over all instances,

$$\epsilon_t = \sum_{i=1}^N D_t(i) \mathbb{I}(h_t(\mathbf{x}_i) \neq y_i). \tag{4}$$

Since the regions S_k form a disjoint partition of data, we can decompose (4) into K regional sums of errors:

$$\epsilon_t = \sum_{k=1}^K \left[\sum_{i \in S_k} D_t(i) \mathbb{I}(h_{t,k}(\mathbf{x}_i) \neq y_i) \right]. \tag{5}$$

Substituting the definition of local error from Eq. (3) into Eq. (5) we get,

$$\epsilon_t = \sum_{k=1}^K W_k \cdot \epsilon_{t,k}. \tag{6}$$

Since $\epsilon_{t,k} < 0.5$ for all k, we have,

$$\epsilon_t < \sum_{k=1}^K W_k \cdot 0.5 = 0.5 \sum_{k=1}^K W_k. \tag{7}$$

Since D_t is a valid probability distribution, $\sum_{k=1}^K W_k = \sum_{i=1}^N D_t(i) = 1$. Therefore,

$$\epsilon_t < 0.5. \tag{8}$$

Thus, the composite learner h_t is a valid weak learner. □

Remark 1 (Theoretical Sufficiency without Router Training): Theorem 2 implies that, from a theoretical standpoint, optimizing the neural network parameters Θ is not required

to guarantee boosting convergence. Convergence only depends on the composite model achieving an error $\epsilon_t < 0.5$. Even a random, untrained projection by the router $\mathcal{R}$ preserves the standard boosting guarantees, provided the local experts can fit their respective random partitions slightly better than random.

Remark 2 (Practical Effectiveness of Router Optimization): While an untrained router theoretically suffices for convergence, training the router parameters is critical for real-world performance. By optimizing the router via gradient descent, the network learns the underlying noise structure of the feature space. This allows it to isolate regions of high noise into specific partitions. Consequently, the local experts assigned to "clean" partitions can achieve a much larger edge ($\epsilon_{t,k} \ll 0.5$), which accelerates overall convergence and prevents the clean base learners from overfitting to regional noise. We empirically validate the necessity of this optimization in Appendix G. Using a 3×3 grid dataset with localized noise, we demonstrate that a trained router outperforms the untrained variants.

5 EXPERIMENTAL RESULTS

We evaluate NRB on both synthetic and real-world datasets. All experiments compare NRB against standard AdaBoost [Freund and Schapire, 1996] and the robust CB-AdaBoost [Xiao et al., 2018] under varying noise levels ($\rho \in \{10\%, 20\%, 30\%\}$). **Reproducibility:** For full reproducibility, including network architectures, optimizer settings, and other hyperparameter values, refer to Appendix B.

5.1 SYNTHETIC VALIDATION

We begin by analyzing the performance of NRB using the simple Gaussian Blob Dataset.

5.1.1 Gaussian Blobs: Robustness at Scale

Setup: We simulate a binary classification problem with overlapping classes sampled from multivariate normal distributions: $\mathcal{N}(\mu_0, I)$ and $\mathcal{N}(\mu_1, I)$ where $\mu_0 = [0, 0]$ and $\mu_1 = [2, 2]$. We inject symmetric label noise at rates $\rho \in \{10\%, 20\%, 30\%\}$. The models are trained on sample sizes ranging from $N = 100$ to $N = 2000$ and evaluated on a clean hold-out set of 10,000 samples. All reported results represent the mean accuracy averaged over $n = 5$ independent trials with different random seeds.

Results: Figure 1 illustrates the learning curves under increasing noise.

- **Low Sample Regime ($N \leq 500$):** At 30% noise, standard AdaBoost collapses to 70.93% accuracy at $N = 100$. In this data-scarce setting, CB-AdaBoost outperforms NRB (81.27% vs. 79.45% at $N = 100$). This aligns with our structural hypothesis: NRB requires sufficient data to optimize the neural router. With limited samples, the router struggles to form stable geometric partitions, making the global statistical weighting of CB-AdaBoost temporarily more effective.
- **High Sample Regime ($N \geq 1000$):** As the sample size increases, the neural router receives enough data to map the noise distribution, allowing NRB to overtake both baselines across all noise levels. At $N = 1000$ with 20% noise, NRB reaches 90.71%. The dominance of NRB continues at $N = 2000$ under 30% noise, where NRB achieves 90.02% accuracy, while CB-AdaBoost begins to stagnate at 89.29%.

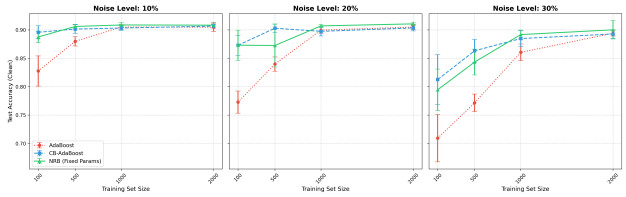

Figure 1: Test accuracy on Gaussian Blobs vs. training size (N)

5.2 DATASETS AND EXPERIMENTAL SETUP

We evaluate our framework on five benchmark datasets from the UCI Machine Learning Repository [Dua and Graff, 2019]. All tasks are formulated as binary classification problems. For multi-class datasets, we apply the following binarization strategies:

- **Isolet (Speech):** The original task is to classify 26 English letters. We convert this into a binary problem by isolating the "E-set" letters (B, C, D, E, G, P, T, V, Z), which are difficult to distinguish due to rhyming, against all other letters.
- **Wine (Tabular):** The original dataset contains three cultivator classes (0, 1, 2). We cast this as a binary detection task for Class 0 (Cultivator A) against the rest (Cultivators B and C).

The characteristics of these datasets are summarized in Table 1. **Noise Injection Protocol:** To simulate label noise, we follow the standard protocol established in robust boosting literature [Xiao et al., 2018]. For a given noise level $\rho \in \{10\%, 20\%, 30\%\}$, we randomly flip the class labels of ρ of the training instances. The test sets are kept clean to evaluate the model's ability to recover the true decision boundary from corrupted training data.

Table 1: Summary of Datasets Used in Experiments. N Denotes the Total Number of Instances, and d Denotes the Feature Dimension.

Dataset	N **(Instances)**	d **(Features)**	**Original Classes**
Small Scale			
Wine	178	13	3
WDBC	569	30	2
Pima	768	8	2
Large Scale			
Isolet	7,797	617	26
Gisette	7,000	5,000	2

5.3 BENCHMARK RESULTS AND ANALYSIS

We compare the classification accuracy of NRB against AdaBoost and CB-AdaBoost. The results for small-scale and large-scale datasets are reported in Table 2 and 3.

Performance on Small Datasets: On small tabular datasets (Wine, WDBC, Pima), NRB consistently outperforms both standard and robust baselines. Standard AdaBoost degrades heavily under noise (e.g., dropping to 73.7% on Wine at 30% noise), whereas NRB maintains a robust 90.0%. NRB surpasses CB-AdaBoost even in low-noise regimes (10% noise) where statistical methods typically dominate, widening this gap under heavy corruption (e.g., 88.65% vs. 86.90% on WDBC at 30% noise). This confirms that the neural router isolates noise pockets even with limited training data, proving to be superior to global statistical weighting.

Performance on High-Dimensional Data: The advantages of NRB are evident on the large-scale datasets (Isolet, Gisette), particularly in high-noise regimes where the feature space is sparse and statistical estimation becomes unstable.

- On **Isolet** ($d = 617$), NRB shows a clear upward trend in relative performance as noise increases. While competitive with CB-AdaBoost at 10% noise, NRB surpasses it at 20% and 30% levels, achieving a peak accuracy of 97.56% (vs. CB-AdaBoost's 97.10%) under severe corruption.
- On **Gisette** ($d = 5000$), the results highlight the limits of global statistical weighting. At lower noise levels (10% and 20%), CB-AdaBoost performs marginally better. However, at 30% noise, NRB overtakes all baselines, maintaining an accuracy of 89.41% while standard AdaBoost collapses to 79.97%. This demonstrates that in very high-dimensional spaces, a structural "divide and conquer" approach offers a stronger defense against heavy noise than global risk estimation.

We benchmarked NRB against a similarly tuned Multi-Layer Perceptron (Appendix F) to confirm that this robustness comes from the routing mechanism rather than inherent neural network capacity. The MLP overfits under high noise (30%), whereas NRB maintains stable boundaries, validating the necessity of structural partitioning.

Extended Benchmark Evaluation: We extended our evaluation to additional datasets under identical noise injection protocols.

Furthermore, we compared our framework against XGBoost [Chen and Guestrin, 2016] across various datasets with varying label noise, verifying that NRB remains competitive against modern ensemble methods.

Due to space constraints, the comprehensive numerical results and comparison plots with XGBoost are detailed in Appendix D and Appendix E, respectively. Consistent with previous results, NRB maintains an advantage, particularly in higher noise levels.

Resolving Spatial Capacity Bottlenecks: To show NRB's structural advantage, we evaluated it on a noisy XOR dataset where global weak learners encounter spatial capacity bottlenecks. As detailed in Appendix H, global baselines struggle to separate non-linear boundaries, often triggering early stopping. In contrast, NRB achieves high accuracy by isolating the quadrants, showing that its routing mechanism provides an advantage in datasets with underlying non-linear structures.

5.4 SENSITIVITY ANALYSIS: EFFECT OF REGION COUNT (K)

The most important hyperparameter of NRB is K, the number of geometric regions the router partitions the space into. We analyze the sensitivity of NRB to K using two experimental setups designed to test the "divide and conquer" hypothesis.

5.4.1 The Grid Dataset Experiment

To isolate the effect of K, we constructed a synthetic 3×3 "Grid Dataset" in $\mathbb{R}^2$ (Figure 2, Left). It contains 9 disjoint geometric regions, each with a unique linear boundary and specific label noise $\rho \in \{0\%, 10\%, 20\%, 30\%\}$ (data generation details in Appendix A.1). This configuration models piecewise linearity and heteroscedastic noise.

To visualize NRB's "divide and conquer" dynamic, we plotted its decision boundary evolution using $K = 9$. Initially, the composite learner captures only a simple global trend (Figure 2, Center). By the end of training (Figure 2, Right), the router compartmentalizes high-noise regions. This allows local experts to recover the true boundaries without distortion.

Table 2: Performance on **Small-Scale** Datasets. The Best Result for Each Setting Is Bolded.

Dataset	Noise	AdaBoost	CB-AdaBoost	NRB (Ours)
Wine	10%	**0.9519 ± 0.0251**	0.9222 ± 0.0474	0.9407 ± 0.0246
	20%	0.8778 ± 0.0557	0.9185 ± 0.0477	**0.9407 ± 0.0216**
	30%	0.7370 ± 0.0872	0.8630 ± 0.0782	**0.9000 ± 0.0800**
WDBC	10%	0.9333 ± 0.0079	0.9263 ± 0.0060	**0.9392 ± 0.0120**
	20%	0.8690 ± 0.0136	0.9170 ± 0.0119	**0.9298 ± 0.0083**
	30%	0.7731 ± 0.0297	0.8690 ± 0.0324	**0.8865 ± 0.0360**
Pima	10%	0.6935 ± 0.0167	0.7368 ± 0.0294	**0.7420 ± 0.0124**
	20%	0.6407 ± 0.0275	0.7273 ± 0.0164	**0.7437 ± 0.0100**
	30%	0.6009 ± 0.0437	0.7074 ± 0.0080	**0.7108 ± 0.0100**

Table 3: Performance on **High-Dimensional** Datasets. The Best Result for Each Setting Is Bolded.

Dataset	Noise	AdaBoost	CB-AdaBoost	NRB (Ours)
Isolet	10%	0.9623 ± 0.0036	**0.9675 ± 0.0049**	0.9670 ± 0.0059
	20%	0.9215 ± 0.0087	0.9662 ± 0.0073	**0.9681 ± 0.0059**
	30%	0.8410 ± 0.0068	0.9710 ± 0.0083	**0.9756 ± 0.0076**
Gisette	10%	0.9133 ± 0.0056	**0.9264 ± 0.0064**	0.9178 ± 0.0065
	20%	0.8700 ± 0.0125	**0.9181 ± 0.0100**	0.9151 ± 0.0066
	30%	0.7997 ± 0.0166	0.8871 ± 0.0142	**0.8941 ± 0.0260**

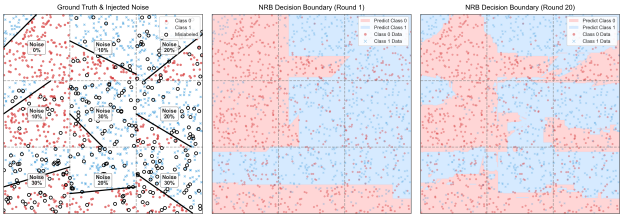

Figure 2: Visualization of the synthetic Grid Dataset and the evolution of the NRB decision boundary ($K = 9$). **(Left)** The true underlying structure, featuring 9 disjoint regions with varying noise levels. **(Center)** The simplistic global boundary learned after a single boosting round. **(Right)** The final composite boundary at the end of training.

To quantify the structural dependency on K, we trained NRB with K varying from 2 to 15. Accuracy rises as K approaches the data's true complexity (Figure 3). Performance peaks at $K = 9$, matching the ground truth. This confirms that NRB effectively utilizes its available buckets to model disjoint boundaries.

5.4.2 Real World Datasets: Regional Noise

To simulate heteroscedastic noise, we adopted a "Four-Quadrant" noise protocol for the Wine, WDBC, and Pima datasets. **Setup:** We generated two random orthogonal hyperplanes centered at the data median, dividing the feature space into four geometric regions. We then injected varying levels of label noise into each region. This creates a complex landscape where the reliability of labels is dependent on the geometric location of the sample. (See Appendix A.2 for more details on noise insertion protocol).

Performance on real-world benchmarks maximizes at $K = 4$ which aligns with the four underlying noise quadrants. This result shows that NRB's "divide and conquer" strategy works, confirming its ability to find and use hidden geometric patterns. Figure 4 shows the test accuracy as a function of K. **In practice**, where the underlying noise distribution is unknown, this demonstrates that K acts as a structural hyperparameter that can be tuned via standard k-fold cross-validation (see Appendix C for supporting experiment).

5.5 ABLATION STUDY: ROUTING DYNAMICS AND COMPONENT ANALYSIS

To isolate the effect of our routing and weighting strategies on model performance, we performed an ablation study on the Gaussian Blobs dataset ($N = 2000$, 30% noise, with other parameters identical to those in Section 5.1.1). We evaluated three distinct routing configurations:

1. **Dynamic Routing (Proposed):** The standard NRB approach where each expert h_t is paired with the specific router state $\mathcal{R}^{(t)}$ active during its training. The final prediction uses the sequence of router snapshots: $F(\mathbf{x}) = \sum \alpha_t h_{t,\mathcal{R}^{(t)}(\mathbf{x})}(\mathbf{x})$.
2. **Static Final:** The router is trained for T rounds, but the *final* state $\mathcal{R}^{(T)}$ is used to route samples for *all* experts in the ensemble, decoupling the experts from their training distributions.
3. **Static Averaged:** Similar to Static Final, but uses a router $\bar{\mathcal{R}}$ whose weights are the arithmetic mean of the router's weights across all T rounds.

We tested these configurations in conjunction with two ensemble weighting methods: **Simple** (standard AdaBoost weight updates) and **Confidence-Based** (CB-AdaBoost weight updates). The results are summarized in Figure 5.

Analysis of Results: This analysis yields three primary observations:

1. Routing consistency dictates stability. Proper dynamic routing is a prerequisite for stability. Static and Averaged variants exhibit reduced accuracy in certain trials (std. dev. $\approx 14\%$), whereas the Dynamic approach remains stable ($< 1.3\%$). This shows that locking the expert to its specific router state is necessary for reliable convergence.

2. Routing dominates Weighting. While Confidence-Based weighting improves the Dynamic model (from 83.9% to 89.8%), it cannot salvage a structurally flawed ensemble. For example, Averaged routing with Confidence weighting

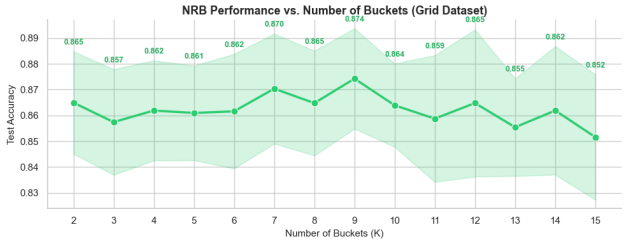

Figure 3: Effect of K on the Grid Dataset. Accuracy peaks at $K = 9$.

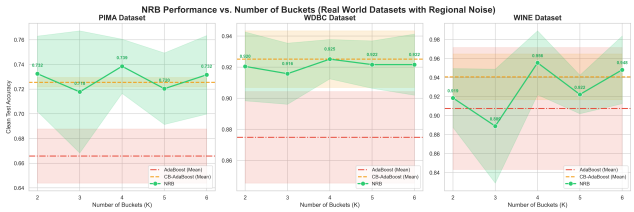

Figure 4: Effect of K on real-world datasets with heteroscedastic noise (4 quadrants).

(75.3%) significantly underperforms Dynamic routing with Simple weighting (83.9%). The geometric decomposition provided by the dynamic router is clearly the primary driver of performance.

3. The Composite Weak Learner Hypothesis. The failure of Static methods validates NRB's premise: the true weak learner is the composite pair $(h_t, \mathcal{R}^{(t)})$. Substituting $\mathcal{R}^{(t)}$ with a future $\mathcal{R}^{(T)}$ or average $\bar{\mathcal{R}}$ forces experts to evaluate distributions unseen during training. This mismatch breaks the ensemble's logic, confirming the router's temporal evolution must be preserved during inference.

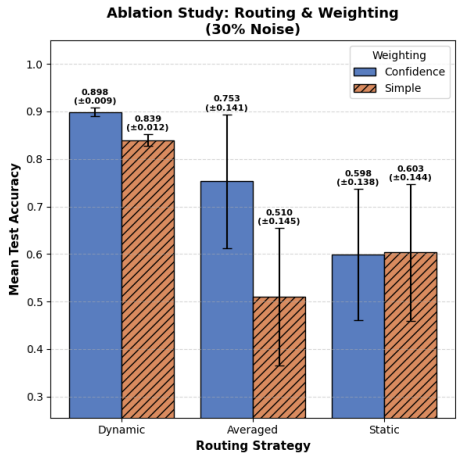

Figure 5: Ablation study on Gaussian Blobs (30% label noise, $N = 2000$). The combination of Neural Routing and Confidence Weighting (proposed NRB algorithm) outperforms other routing and weighting strategies.

6 DISCUSSION AND LIMITATIONS

While robust against heteroscedastic noise, NRB introduces trade-offs in computational complexity and interpretability. **Computational Complexity:** Both NRB and CB-AdaBoost incur a one-time initialization cost of $\mathcal{O}(N \log N \cdot d)$ to estimate label confidence via k-nearest neighbor search (assuming a kd-tree implementation). AdaBoost avoids this overhead, initializing with uniform weights in $\mathcal{O}(N)$. In the iterative phase, AdaBoost and CB-AdaBoost have a complexity of $\mathcal{O}(TNd)$, for decision tree training. NRB requires additional per-round compute to generate router targets via the K experts and to train the neural router itself. For E epochs of training a router of hidden width h, the total iterative complexity approximates:

$$\mathcal{O}(T(Nd + NC_{\text{inf}} + ENh)), \tag{9}$$

where C_{inf} represents the negligible inference cost of the weak learners. However, faster convergence amortizes this increased per-round cost. On Gisette (Table 3), NRB achieves superior accuracy in just $T = 10$ rounds, whereas standard baselines require $T \geq 50$ for asymptotic performance. Thus, NRB trades higher per-iteration compute for fewer total boosting rounds.

Limitations: First, NRB is sensitive to the bucket count K (Section 5.4). A low K leaves noisy and clean regions mixed (under-partitioning), while a high K over-fragments data, depriving local experts of sufficient samples. **Second,** the neural router reduces overall interpretability of the model. Unlike standard tree ensembles, NRB's decision boundary is composed of a non-linear neural projection with local trees, complicating feature importance attribution.

7 CONCLUSION

In this paper, we introduced Neural Routed Boosting (NRB), an ensemble framework designed to handle heteroscedastic label noise. Unlike methods relying on global statistical approximations, NRB employs a "divide and conquer" strategy. It decouples spatial partitioning from classification by using a Neural Router to isolate noisy regions for localized experts. We proved that this composite model guarantees training convergence. NRB demonstrated significant robustness, particularly in high-noise and high-dimensional settings, and recovered underlying noise structures during sensitivity analysis. Future work will explore end-to-end differentiable routing and multi-class extensions.

8 BROADER IMPACT

NRB improves model reliability in critical domains with imperfect data, such as medical diagnosis. However, the routing mechanism introduces algorithmic fairness risks. The network might inadvertently partition data using protected attributes (e.g., race or gender) to easily separate clean and noisy samples, leading to disparate impact. Practitioners must audit the demographics of the learned partitions to prevent discriminatory splits.

References

Stefan Aeberhard and M. Forina. Wine. UCI Machine Learning Repository, 1992. DOI: https://doi.org/10.24432/C5PC7J.

Leo Breiman and Charles J. Stone. Waveform Database Generator (Version 1). UCI Machine Learning Repository, 1984. DOI: https://doi.org/10.24432/C5CS3C.

Leo Breiman, Jerome Friedman, Charles J. Stone, and Richard A. Olshen. *Classification and regression trees*. Wadsworth International Group, 1984.

Jian Cao, Sam Kwong, and Ruliang Wang. A noise-detection based AdaBoost algorithm for mislabeled data. *Pattern Recognition*, 45(12):4451–4465, 2012.

David Chapman and Ajay Jain. Musk (Version 2). UCI Machine Learning Repository, 1994. DOI: https://doi.org/10.24432/C51608.

Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 785–794. ACM, 2016.

Ron Cole and Mark Fanty. ISOLET. UCI Machine Learning Repository, 1991. DOI: https://doi.org/10.24432/C51G69.

Thomas G. Dietterich. An experimental comparison of three methods for constructing ensembles of decision trees: Bagging, boosting, and randomization. *Machine Learning*, 40(2):139–157, 2000.

Carlos Domingo and Osamu Watanabe. MadaBoost: A modification of AdaBoost. In *Proceedings of the Thirteenth Annual Conference on Computational Learning Theory (COLT)*, pages 180–189. Morgan Kaufmann, 2000.

Dheeru Dua and Casey Graff. UCI machine learning repository. University of California, Irvine, School of Information and Computer Sciences, 2019. URL: http://archive.ics.uci.edu/ml.

Yoav Freund. An adaptive version of the boost by majority algorithm. *Machine Learning*, 43(3):293–318, 2001.

Yoav Freund and Robert E. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning (ICML)*, pages 148–156. Morgan Kaufmann, 1996.

Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting. *The Annals of Statistics*, 28(2):337–407, 2000.

Yuning Gao and Fuqing Gao. Edited AdaBoost by weighted kNN. *Neurocomputing*, 73(16–18):3079–3088, 2010.

B. German and Vina Spiehler. Glass Identification. UCI Machine Learning Repository, 1987. DOI: https://doi.org/10.24432/C5WW2P.

Isabelle Guyon, Steve Gunn, Asa Ben-Hur, and Gideon Dror. Gisette. UCI Machine Learning Repository, 2004. DOI: https://doi.org/10.24432/C5HP5B.

Robert A. Jacobs, Michael I. Jordan, Steven J. Nowlan, and Geoffrey E. Hinton. Adaptive mixtures of local experts. *Neural Computation*, 3(1):79–87, 1991.

Volker Lohweg. Banknote Authentication. UCI Machine Learning Repository, 2012. DOI: https://doi.org/10.24432/C55P57.

Kenta Nakai. Ecoli. UCI Machine Learning Repository, 1996. DOI: https://doi.org/10.24432/C5388M.

Rocco A. Servedio. Smooth boosting and learning with malicious noise. *Journal of Machine Learning Research*, 4:633–648, 2003.

Jack W. Smith, J. E. Everhart, W. C. Dickson, W. C. Knowler, and R. S. Johannes. Using the ADAP learning algorithm to forecast the onset of diabetes mellitus. In *Proceedings of the Annual Symposium on Computer Application in Medical Care*, pages 261–265. American Medical Informatics Association, 1988.

Alexander Vezhnevets and Vladimir Vezhnevets. Modest AdaBoost - teaching AdaBoost to generalize better. In *Graphicon*, pages 987–997. Graphicon, 2005.

William H. Wolberg and Olvi L. Mangasarian. Breast Cancer Wisconsin (Original). UCI Machine Learning Repository, 1990. DOI: https://doi.org/10.24432/C5HP4Z.

William H. Wolberg, Olvi L. Mangasarian, and W. Nick Street. Breast Cancer Wisconsin (Diagnostic). UCI Machine Learning Repository, 1993. DOI: https://doi.org/10.24432/C5DW2B.

William H. Wolberg, W. Nick Street, and Olvi L. Mangasarian. Breast Cancer Wisconsin (Prognostic). UCI Machine Learning Repository, 1995. DOI: https://doi.org/10.24432/C5GK50.

Zhi Xiao, Zhe Luo, Bo Zhong, and Xin Dang. Robust and efficient boosting method using the conditional risk. *IEEE Transactions on Neural Networks and Learning Systems*, 29(7):3069–3083, 2018.

Chuan-Xian Zhang and Jian-Sheng Zhang. A local boosting algorithm for solving classification problems. *Computational Statistics & Data Analysis*, 52(4):1928–1941, 2008.

Chuan-Xian Zhang, Jian-Sheng Zhang, and Guo-Ying Zhang. An efficient modified boosting method for solving classification problems. *Journal of Computational and Applied Mathematics*, 214(2):381–392, 2008.

Neural Routed Boosting: Robust Learning against Heteroscedastic Noise (Supplementary Material)

Puspak Chakraborty[1] **Arun Rajkumar**[1]

[1]Data Science and Artificial Intelligence Dept., Indian Institute of Technology, Chennai, India

A DATASET GENERATION AND PROTOCOLS

A.1 SYNTHETIC GRID DATASET GENERATION

To evaluate the capabilities of Neural Routed Boosting (NRB), we constructed a synthetic dataset that models a piecewise linear decision boundary with varying noise levels (Section 5.4.1).

- **Structure:** The input space $\mathbb{R}^2$ is partitioned into a 3×3 grid, creating 9 disjoint geometric regions. The dataset consists of 1000 training samples and 5000 test samples.
- **Local Linearity & Heteroscedastic Noise:** Each region (r, c) is assigned a unique linear decision boundary defined by $y > mx + c$ (in local coordinates) and a specific label noise probability ρ. The complete set of parameters for all 9 regions is detailed in Table 4. This configuration ensures that no single global linear classifier can solve the problem, and that the reliability of labels is location-dependent.
- **Visualization:** A generated sample of this dataset, visualizing the decision boundaries and noise distribution, is provided in Figure 2.

Table 4: Exact Parameters for the Synthetic Grid Dataset. "Row 0" Corresponds to the Top of the Grid. Local Coordinates Are Normalized to $[0, 1]$ Within Each Region.

Grid Position	Region	Slope (m)	Intercept (c)	Noise (ρ)
Top Row				
Top-Left	(0,0)	1.0	0.5	0%
Top-Mid	(0,1)	-0.5	0.6	10%
Top-Right	(0,2)	0.8	-0.1	20%
Middle Row				
Mid-Left	(1,0)	0.7	0.5	10%
Center	(1,1)	-1.0	0.5	30%
Mid-Right	(1,2)	-0.6	0.5	20%
Bottom Row				
Bot-Left	(2,0)	0.3	0.4	30%
Bot-Mid	(2,1)	0.1	0.3	20%
Bot-Right	(2,2)	-0.7	0.6	30%

A.2 HETEROSCEDASTIC NOISE INSERTION PROTOCOL (FOUR-QUADRANT)

For the "Real World Datasets" experiment (Section 5.4.2), we simulated heteroscedastic noise by partitioning the high-dimensional feature space into four quadrants using random hyperplanes. Let $X \in \mathbb{R}^{N \times d}$ be the centered input data. We generate two random orthogonal vectors $\mathbf{w}_1, \mathbf{w}_2 \in \mathbb{R}^d$ and define two separating hyperplanes passing through the median of the data projections:

$$h_1(\mathbf{x}) = \mathbf{w}_1^\top \mathbf{x} + b_1, \quad h_2(\mathbf{x}) = \mathbf{w}_2^\top \mathbf{x} + b_2, \tag{10}$$

where biases b_1, b_2 are calibrated such that each plane splits the data median (ensuring roughly balanced regions). The space is then divided into four regions based on the sign of these projections. Table 5 details the definition and the noise level (ρ) injected into each quadrant. This protocol ensures that the noise level is deterministic with respect to the input features $\mathbf{x}$ but statistically independent of the true label y, simulating sensor-specific or region-specific failure modes (e.g., a sensor failing only when temperature and pressure are both high).

Table 5: Noise Allocation for the Four-Quadrant Experiment. Regions Are Defined by the Sign of the Projection onto the Two Random Hyperplanes h_1 and h_2.

Region ID	Geometric Condition	Description	Noise (ρ)
Region 0	$h_1(\mathbf{x}) < 0 \wedge h_2(\mathbf{x}) < 0$	Bottom-Left Quadrant	10%
Region 1	$h_1(\mathbf{x}) \geq 0 \wedge h_2(\mathbf{x}) < 0$	Bottom-Right Quadrant	20%
Region 2	$h_1(\mathbf{x}) < 0 \wedge h_2(\mathbf{x}) \geq 0$	Top-Left Quadrant	30%
Region 3	$h_1(\mathbf{x}) \geq 0 \wedge h_2(\mathbf{x}) \geq 0$	Top-Right Quadrant	15%

B HYPERPARAMETER CONFIGURATION

In all experimental settings, the neural router was optimized using the Adam algorithm, and ReLU was used as non-linearity. **Evaluation and Tuning Protocol:** To evaluate robustness, we adhered to a consistent data splitting protocol across all datasets: 30% of the data was held out as a **clean test set**, while the remaining 70% was subjected to label noise injection and utilized for training. The hyperparameter selection strategy was adapted to the dataset scale:

- **Small Datasets (Wine [Aeberhard and Forina, 1992], WDBC [Wolberg et al., 1993], Pima [Smith et al., 1988]):** Optimal hyperparameters were identified via **3-fold cross-validation** performed on the noisy training partition (70%).
- **High-Dimensional Datasets (Isolet [Cole and Fanty, 1991], Gisette [Guyon et al., 2004]):** A fixed 10% **validation split** was derived from the noisy training data to guide the tuning process.

All reported performance metrics represent the mean and standard deviation averaged over $N = 5$ independent trials, utilizing distinct random seeds for data partitioning and noise injection. **Ablation Study Parameter Configuration:** To isolate the impact of individual components, we maintained a fixed hyperparameter configuration across all evaluated variants:

- **Boosting:** $T = 20$ boosting rounds.
- **Confidence Estimation:** Neighborhood size $k_{nn} = 10$.
- **Neural Router:** $K = 4$ buckets, Hidden layers $[16, 8]$, trained for 50 epochs per round (batch size 64, learning rate 0.01).

B.1 NOTE ON CONFIGURATION FOR HIGH-DIMENSIONAL DATASETS

Due to the high dimensionality of Isolet ($d = 617$) and the extreme dimensionality of Gisette ($d = 5000$), running standard AdaBoost and CB-AdaBoost with the default 200 estimators was computationally prohibitive across multiple noise trials. To ensure feasible training times, we adjusted the baseline capacities as follows:

- **Isolet:** The number of estimators for AdaBoost and CB-AdaBoost was reduced to $T = 100$.
- **Gisette:** The number of estimators was further reduced to $T = 50$. Additionally, the neighborhood size for risk estimation was reduced to $k_{nn} = 5$.

Table 6: Fixed Parameters for All Experiments.

Parameter	Gaussian Blobs	Small Datasets	Isolet	Gisette	Grid Dataset	Real-World (Regional Noise)
Weak Learner (All Models)						
Base Algorithm	CART	CART	CART	CART	CART	CART
Max Depth	3	3	3	3	3	3
Baselines						
Estimators	100	200	100	50	-	-
Neighbors (k_{nn})	5	10	10	5	-	-
NRB Fixed Settings						
Rounds (T)	10	20	15	10	20	20
Buckets (K)	**2**	**2**	**2**	**2**	**2-15**	**2-6**
Hidden Layers	[8,4]	Tuned (Table 7)	[32]	[16]	[16,8]	Tuned (Table 7)
Router k_{nn}	5	10	10	5	15	10
Batch Size	512	64	2048	4096	512	Tuned (Table 7)
Epochs	20	10	Tuned (Table 7)	5	50	Tuned (Table 7)
Router LR	0.1	Tuned (Table 7)	0.05	0.015	0.01	Tuned (Table 7)

Table 7: Hyperparameter Search Space for the Neural Router (NRB).

Hyperparameter	Small Datasets	Isolet	Real-World (Regional Noise)
Hidden Layers	$\{[8], [16, 8]\}$	Fixed (Table 6)	$\{[8, 4], [16, 8]\}$
Batch Size	Fixed (Table 6)	Fixed (Table 6)	$\{32, 64, 128\}$
Epochs	Fixed (Table 6)	$\{5, 10\}$	$\{30, 50\}$
Router LR	$\{0.01, 0.001\}$	Fixed (Table 6)	$\{0.01, 0.05\}$

C HYPERPARAMETER SELECTION STRATEGY FOR K

As discussed in Section 5.4.2, the number of buckets K acts as a structural hyperparameter that aligns the model with the underlying noise distribution. By matching the number of buckets to the complexity of the noise distribution, the model can isolate corrupted regions without unnecessarily fragmenting the feature space. In practical applications where this noise distribution is unknown, K must be selected either by cross-validation or domain knowledge. To validate this, we conducted a 5-fold cross-validation experiment on three small-scale datasets (Pima, WDBC, Wine). We injected synthetic noise based on the 4-region orthogonal map described in A.2 (implying an optimal ground truth of $K \approx 4$). We then evaluated candidate values $K \in \{3, 4, 5\}$ while fixing all other hyperparameters. These candidate values were deliberately chosen to evaluate the algorithm's behavior under both under-segmentation ($K = 3$) and over-segmentation ($K = 5$) conditions relative to the true generative process. The results, summarized in Table 8, demonstrate that standard cross-validation frequently identifies the optimal $K = 4$.

Table 8: Frequency of Winning K Values Across 5 Independent Trials (5-Fold CV per Trial). The Noise Structure Was Generated Using 4 Orthogonal Regions (See Appendix A.2).

Dataset	Winner: K=3	Winner: K=4	Winner: K=5	Dominant Mode
Pima	3	1	1	K=3 (Under-segmentation)
WDBC	0	3	2	**K=4 (Optimal)**
Wine	1	3	1	**K=4 (Optimal)**

Experimental Configuration:

For each of the $N = 5$ trials, the dataset was stratified and split into a 30% **clean test set** (for final evaluation) and a

Table 9: Fixed Hyperparameters for the Cross-Validation Experiment on Pima, WDBC, and Wine Datasets.

Component	Parameter Setting
Weak Learner	Algorithm: CART Decision Tree Max Depth: 3
Neural Router	Hidden Layers: [16, 8] (ReLU) Optimization: Adam ($\eta = 0.05$) Training: 30 Epochs/Round, Batch Size 64
Ensemble	Rounds (T): 20 Neighbors (k_{nn}): 10

70% **noisy training set**. The reported 5-fold cross-validation was performed on the noisy training partition to simulate a realistic deployment scenario where clean ground truth labels are unavailable during hyperparameter tuning. This separation guarantees that our model selection process relies on corrupted validation data, providing a rigorous assessment of the algorithm's practical utility.

D EXTENDED BENCHMARK EVALUATION

To validate the robustness of Neural Routed Boosting against a wider variety of datasets, we included seven datasets from the UCI Machine Learning Repository [Dua and Graff, 2019]. These datasets span diverse domains, including medical diagnostics (Breast Cancer [Wolberg and Mangasarian, 1990], WPBC [Wolberg et al., 1995]), physical properties (Glass [German and Spiehler, 1987]), biological classification (Ecoli [Nakai, 1996]), financial verification (Banknote [Lohweg, 2012]), signal processing (Waveform [Breiman and Stone, 1984]), and molecular biology (Musk [Chapman and Jain, 1994]).

Consistent with the experimental protocol described in Section 5.2, we evaluated standard AdaBoost, CB-AdaBoost, and NRB under label noise levels of $\rho \in \{10\%, 20\%, 30\%\}$. For these experiments, the hyperparameter configurations across all models were kept same as the hyperparameters used for Small Datasets in Table 6. The results, presented in Table 11, demonstrate that NRB achieves competitive or superior performance, particularly in the high-noise (30%) regime.

Dataset Characteristics and Binarization: Table 10 summarizes the characteristics of the seven additional datasets. For multi-class datasets (Glass, Ecoli, and Waveform), we applied One-vs-Rest binarization. The first class is labeled as the positive class ($y = 1$), while all remaining classes are grouped as the negative class ($y = 0$).

Table 10: Summary of Extended Datasets. N denotes the total number of instances, and d denotes the original feature dimension.

Dataset	N **(Instances)**	d **(Features)**	**Original Classes**
Breast Cancer	699	9	2
WPBC	198	33	2
Banknote	1,372	4	2
Musk	6,598	166	2
Multi-class (Binarized via One-vs-Rest)			
Glass	214	9	6
Ecoli	336	7	8
Waveform	5,000	21	3

E COMPARISON WITH XGBOOST

To ensure our framework remains competitive against modern, state-of-the-art ensemble methods, we benchmarked Neural Routed Boosting against XGBoost [Chen and Guestrin, 2016]. This evaluation was conducted in accordance with the experimental protocol and noise injection procedures detailed in Section 5.2.

As shown in Figure 6, XGBoost performs well in lower noise setups, but its performance degrades when high label noise is introduced. In contrast, NRB maintains a comparatively steady performance across all noise settings.

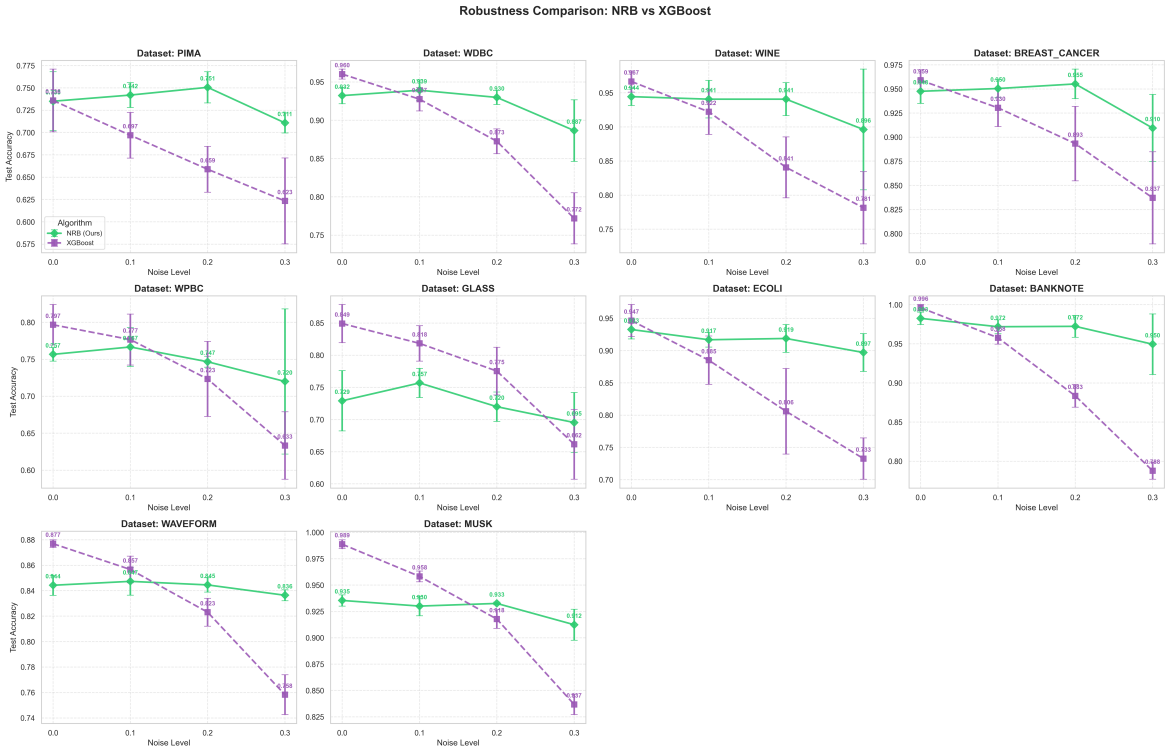

Figure 6: Robustness comparison between NRB and XGBoost across varying noise levels. While the XGBoost performs well in lower noise, but NRB consistently outperforms XGBoost in higher noise settings.

F COMPARISON WITH MULTI-LAYER PERCEPTRON (MLP)

To further validate the necessity of NRB's routed architecture, we conducted an additional experiment comparing our framework against a standard Multi-Layer Perceptron (MLP). The objective of this experiment was to determine whether a standard neural network, given the same representational capacity as the NRB router, could learn to ignore the noise and not overfit on the training data.

F.1 EXPERIMENTAL SETUP

We evaluated both models on the small-scale tabular datasets (Wine, WDBC, Pima) under symmetric label noise levels of $\rho \in \{10\%, 20\%, 30\%\}$.

- **Data Partitioning:** For each of the $N = 5$ independent trials, the data was stratified and split into a 70% noisy training set and a 30% clean test set.
- **Hyperparameter Tuning:** To ensure a fair comparison, both the MLP and the NRB neural router were subjected to the same hyperparameter search space. A 3-fold cross-validation was performed on the corrupted training partition.

Model Configurations: The hyperparameters for both architectures are detailed in Table 12.

F.2 RESULTS AND ANALYSIS

The results of the comparative analysis are visualized in Figure 7. While the MLP demonstrates reasonable capability at the lowest noise level (10%), its performance rapidly deteriorates as the corruption rate increases. At 30% label noise, the

Table 11: Performance on Expanded Datasets under varying noise levels.

Dataset	Noise	AdaBoost	CB-AdaBoost	NRB (Ours)
Breast Cancer	10%	0.9295 ± 0.0092	0.9448 ± 0.0175	$\mathbf{0.9495 \pm 0.0093}$
	20%	0.8867 ± 0.0285	0.9314 ± 0.0197	$\mathbf{0.9505 \pm 0.0111}$
	30%	0.8419 ± 0.0273	0.9114 ± 0.0285	$\mathbf{0.9162 \pm 0.0275}$
WPBC	10%	$\mathbf{0.7800 \pm 0.0356}$	0.7667 ± 0.0258	0.7667 ± 0.0236
	20%	0.7167 ± 0.0548	$\mathbf{0.7567 \pm 0.0133}$	0.7467 ± 0.0067
	30%	0.6567 ± 0.0170	0.7033 ± 0.0785	$\mathbf{0.7200 \pm 0.0878}$
Glass	10%	$\mathbf{0.8308 \pm 0.0487}$	0.7846 ± 0.0424	0.7569 ± 0.0204
	20%	$\mathbf{0.7631 \pm 0.0359}$	0.7169 ± 0.0473	0.7200 ± 0.0204
	30%	0.6615 ± 0.0550	$\mathbf{0.6954 \pm 0.0711}$	$\mathbf{0.6954 \pm 0.0417}$
Ecoli	10%	0.8653 ± 0.0418	$\mathbf{0.9228 \pm 0.0074}$	0.9168 ± 0.0101
	20%	0.8020 ± 0.0300	0.9069 ± 0.0389	$\mathbf{0.9188 \pm 0.0192}$
	30%	0.7307 ± 0.0373	0.8931 ± 0.0510	$\mathbf{0.8970 \pm 0.0263}$
Banknote	10%	0.9743 ± 0.0088	0.9631 ± 0.0039	$\mathbf{0.9748 \pm 0.0103}$
	20%	0.9316 ± 0.0100	0.9670 ± 0.0079	$\mathbf{0.9723 \pm 0.0124}$
	30%	0.8976 ± 0.0130	$\mathbf{0.9515 \pm 0.0208}$	0.9495 ± 0.0346
Waveform	10%	0.8456 ± 0.0104	0.8379 ± 0.0085	$\mathbf{0.8473 \pm 0.0098}$
	20%	0.8216 ± 0.0085	0.8287 ± 0.0122	$\mathbf{0.8445 \pm 0.0051}$
	30%	0.7819 ± 0.0138	0.8336 ± 0.0081	$\mathbf{0.8364 \pm 0.0039}$
Musk	10%	$\mathbf{0.9439 \pm 0.0077}$	0.9159 ± 0.0167	0.9301 ± 0.0083
	20%	0.9280 ± 0.0041	0.9200 ± 0.0074	$\mathbf{0.9326 \pm 0.0011}$
	30%	0.8999 ± 0.0107	0.9015 ± 0.0130	$\mathbf{0.9124 \pm 0.0131}$

MLP suffers from overfitting across all three datasets. Because the standard MLP optimizes a global loss function without any structural mechanism to isolate corrupted regions, it memorizes the noisy training labels. In contrast, NRB maintains a robust performance curve. The "divide and conquer" routing mechanism prevents localized noise pockets from destabilizing the clean geometric regions. These results confirm that NRB's structural partitioning provides a superior defense against label noise compared to the implicit feature transformations of a standard neural network.

G IMPACT OF NEURAL ROUTER OPTIMIZATION

As discussed in Remark 2 (Section 4), Theorem 2 provides a sufficiency proof for convergence even if the neural router is not optimized. To empirically demonstrate that gradient-based router optimization is essential for practical performance, we conducted the following experiment.

In this experiment, we compared our optimized Neural Router against two unoptimized baselines:

1. **Fixed Random NN:** A randomly initialized router whose static partition is applied across all boosting rounds.
2. **Variable Random NN:** A randomly initialized router that is re-randomized at every boosting iteration.
3. **Trained NRB (Ours):** Our standard framework where the router is optimized via gradient descent.

Data Generation Protocol: 3×3 **Grid**
To evaluate the algorithm's robustness against spatially varying corruption, we utilized the synthetic 3×3 Grid Dataset (introduced in Appendix A.1). The feature space $\mathbf{x} \in \mathbb{R}^2$ is uniformly partitioned into 9 distinct spatial regions, $S_0, \ldots, S_8$.

Table 12: Fixed and Tuned Hyperparameters for the NRB vs. MLP Comparison Experiment.

Component	Parameter Setting
Common Parameters	Activation: ReLU Optimization: Adam ($\eta = 0.01$) Batch Size: 64
MLP	Hidden Layers: Tuned $\in \{[8], [16, 8]\}$ Epochs: 200
NRB	Router Hidden Layers: Tuned $\in \{[8], [16, 8]\}$ Boosting Rounds (T): 20 Neural Network Epochs per Boosting Round: 10 Buckets (K): 2 Weak Learner: CART (Max Depth = 3)

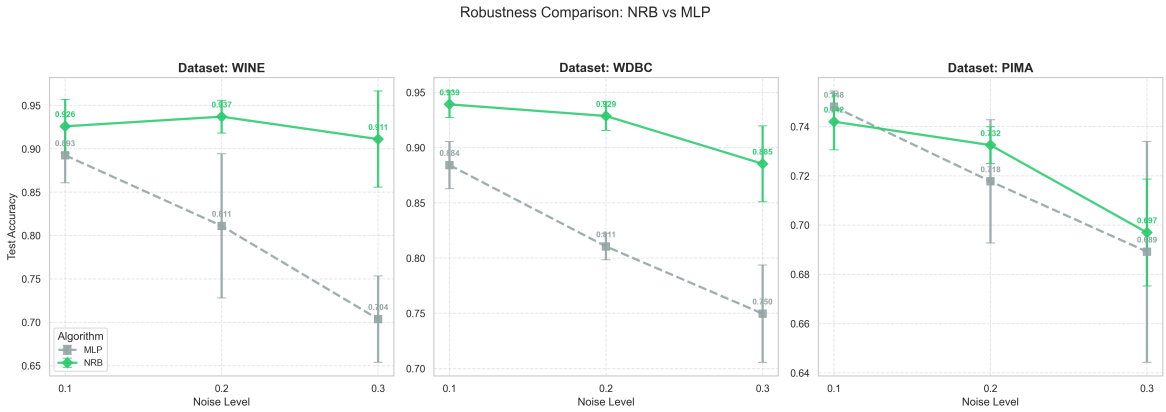

Figure 7: Robustness comparison between NRB and MLP across varying noise levels. While the MLP overfits at higher noise regimes (30%), NRB relies on its routing mechanism to isolate corrupted samples, maintaining higher test accuracy.

- **Local Decision Boundaries:** Based upon the parameters detailed in Table 4, the true underlying class label $y^* \in \{0, 1\}$ for a sample $\mathbf{x} \in S_k$ is determined by a region-specific linear boundary:

$$y^* = \mathbb{I}(x_2 > m_k x_1 + c_k) \tag{11}$$

  where m_k and c_k are the predefined slope and intercept for region k and $\mathbb{I}(\cdot)$ denotes the indicator function.

- **Heteroscedastic Noise Injection:** We apply a spatially varying label noise function $\eta(\mathbf{x}) = P(y \neq y^* \mid \mathbf{x} \in S_k)$. The noise rate for each region is dictated by the base spatial pattern matrix M, which is scaled by a peak noise parameter $p \in \{0.0, 0.1, 0.2, 0.3\}$:

$$M = \begin{bmatrix} 0.00 & 0.33 & 0.66 \\ 0.33 & 1.00 & 0.66 \\ 1.00 & 0.66 & 1.00 \end{bmatrix}, \quad \eta_k = p \cdot M_{i,j} \tag{12}$$

  For example, the top-left region always acts as a perfectly clean region (0% noise multiplier), while the center, bottom-left and bottom-right regions act as heavily corrupted regions (100% of the peak noise level p).

- **Evaluation:** To measure boundary recovery, algorithms are trained on the corrupted distribution but evaluated on noiseless test set ($p = 0.0$).

Results and Analysis:
As illustrated in Figure 8, the trained variant performs better than the other variants across all noise levels. This empirically shows that an optimized router is able to better represent the underlying noise pattern than its un-optimized counterparts.

Experiment Details: The configurations for this experiment are detailed in Table 13.

Table 13: Hyperparameter Configuration for the Router Training Ablation Study (3×3 Grid Dataset).

Component	Parameter Setting
Data Generation	Training Samples: 2000 Testing Samples: 10000 Grid Dimension: 3×3 (9 distinct regions)
Common NRB Parameters	Boosting Rounds (T): 20 Buckets (K): 9 Weak Learner: CART (Max Depth = 3) Confidence Estimation (k_{nn}): 10 Weighting Strategy: Confidence-based
Neural Router	Hidden Layers: [16, 8] (ReLU) Optimization: Adam ($\eta = 0.01$) Batch Size: 64
Router Optimization	**Trained NRB (Ours):** 50 Epochs/Round

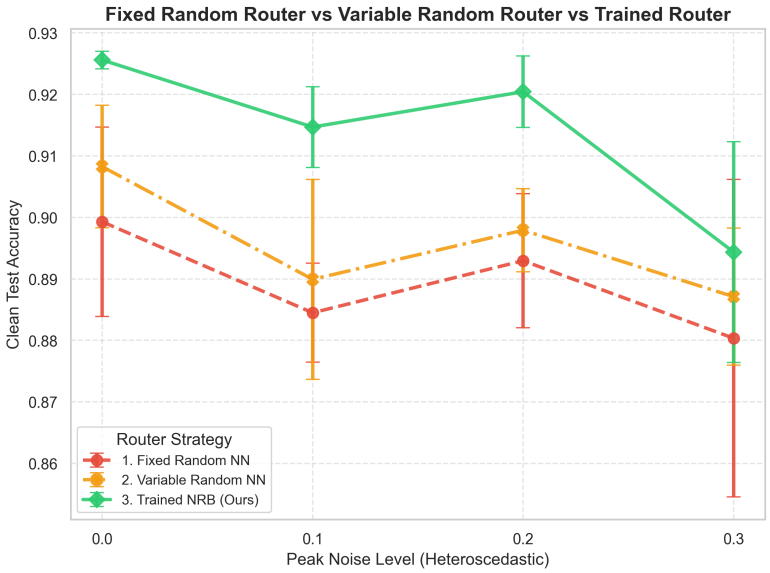

Figure 8: Impact of Neural Router Training on the 3×3 Grid Dataset under varying peak noise levels (p).

H RESOLVING SPATIAL CAPACITY BOTTLENECKS: THE NOISY XOR DATASET

Data Generation Protocol:

In order to evaluate the algorithms' capacity to resolve non-linear boundaries under spatially varying corruption, we construct a synthetic dataset based on the exclusive-OR (XOR) problem. A sample dataset is given in Figure 9.

- **Base Geometry & True Labels:** The feature space consists of four distinct Gaussian clusters positioned in the four quadrants of a 2D plane. The centroids are located at $\mu_1 = (2, 2)$, $\mu_2 = (-2, -2)$, $\mu_3 = (-2, 2)$, and $\mu_4 = (2, -2)$. We assign binary labels $y^* \in \{0, 1\}$ in a cross-diagonal pattern to create a non-linear XOR manifold:

$$y^* = 1 \quad \text{for clusters at } \mu_1 \text{ and } \mu_2$$
$$y^* = 0 \quad \text{for clusters at } \mu_3 \text{ and } \mu_4$$

  This geometry guarantees that a single global linear hyperplane (e.g., a depth-1 decision stump) cannot successfully separate the classes.
- **Heteroscedastic Noise Injection:** We inject label noise unevenly across the feature space to test robustness against

localized corruption. We apply a region-specific noise rate $\eta_k = P(y \neq y^* \mid \mathbf{x} \in \text{Cluster}_k)$ using the noise array $\eta = [0.3, 0.0, 0.3, 0.0]$ corresponding to the four centroids. This means 30% of the samples in the top-right (μ_1) and top-left (μ_3) clusters have their labels randomly flipped, while the bottom-left and bottom-right clusters receive 0% noise.

- **Train and Test Sets:** The training set is generated with 1,000 samples per cluster (4,000 total) and includes the heteroscedastic noise. The testing set is generated with 2,000 samples per cluster (8,000 total) but is left noise-free. This ensures that the evaluation metric measures how well the algorithms discovered the underlying XOR boundaries, completely ignoring the noise on which they were trained.

The Model Configuration (The "Bottleneck"):

The experiment is set up to run across 5 trials, comparing three algorithms: standard AdaBoost, CB-AdaBoost, and Neural Routed Boosting (NRB).

- **The Weak Learner:** Crucially, all three algorithms are strictly forced to use Depth-1 Decision Trees.
- **The Constraint:** A Depth-1 Decision Tree can only draw a single horizontal or vertical line. In this data setup any single straight line will group a Class 1 cluster with a Class 0 cluster, guaranteeing an initial error rate of roughly 50%.
- **NRB Configuration:** NRB is provisioned with $K = 4$ buckets. This allows the neural router to slice the dataset into 4 subspaces before passing the data to the decision stumps.

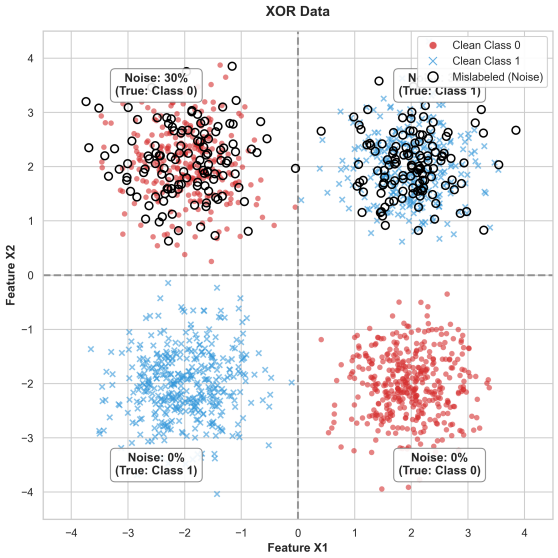

Figure 9: Visualization of the Noisy XOR manifold. The top clusters contain 30% label noise, while the bottom clusters are clean.

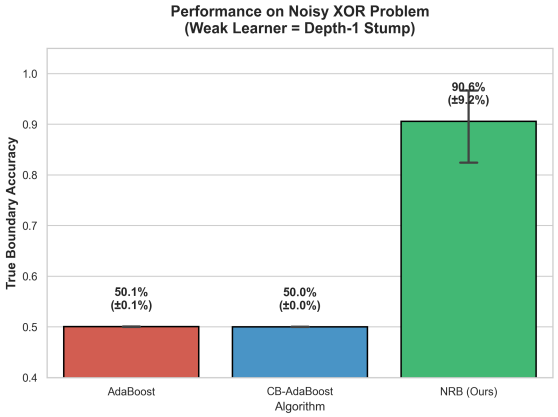

Figure 10: Accuracy results. NRB perfectly isolates the quadrants, while global baselines collapse under the spatial bottleneck.

As shown in Figure 10, standard AdaBoost and CB-AdaBoost fail to resolve the underlying boundary. In contrast, NRB successfully bypasses this structural bottleneck. By routing the data into four geometric buckets, the router assigns each XOR cluster its own local expert. This transforms the global non-linear problem into four trivial, linearly separable tasks, allowing NRB to recover the true decision boundaries despite the heavy localized noise. This shows that dynamic geometric routing provides a structural advantage over global boosting algorithms when navigating complex, non-linear feature spaces.

Experiment Details: The configurations for this experiment are detailed in Table 14.

I STANDARD ADABOOST ALGORITHM

For completeness, we provide the standard AdaBoost algorithm (Algorithm 3) as formulated by Freund and Schapire [1996].

Table 14: Hyperparameter Configuration for the Spatial Capacity Bottleneck Experiment.

Component	Parameter Setting
Data Generation	Training Samples: 4,000 (1,000 per cluster) Testing Samples: 8,000 (2,000 per cluster) Geometry: 4 Gaussian Clusters (XOR logic) Noise Distribution: Heteroscedastic ($\rho \in \{30\%, 0\%, 30\%, 0\%\}$)
Global Constraint	Weak Learner: CART Decision Stump (Max Depth = 1)
Baselines	AdaBoost Rounds (T): 10 CB-AdaBoost Rounds (T): 10 CB-AdaBoost Confidence Estimation (k_{nn}): 15
NRB (Ours)	Boosting Rounds (T): 20 Buckets (K): 4 Weighting Strategy: Confidence-based ($k_{nn} = 15$) Routing Mode: Dynamic
Neural Router	Hidden Layers: [16, 8] (ReLU) Optimization: Adam ($\eta = 0.01$) Training: 50 Epochs/Round Batch Size: 512

Algorithm 3: Standard AdaBoost Algorithm

Input: Training set $S = \{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$, iterations T.
Initialization:
for *all* $i \in \{1, \dots, N\}$ **do**
  $D_i^{(1)} \leftarrow \frac{1}{N}$
end
for $t = 1$ **to** T **do**
  `// 1. Train Weak Learner`
  Train weak learner h_t on S using data distribution $D^{(t)}$
  `// 2. Compute Error`
  $\epsilon_t \leftarrow \sum_{i=1}^{N} D_i^{(t)} \mathbb{I}(h_t(\mathbf{x}_i) \neq y_i)$
  `// 3. Calculate Voting Weight`
  $\alpha_t \leftarrow \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
  if $\alpha_t < 0$ **then**
    break (Learner worse than random guessing)
  end
  `// 4. Update Data Distribution`
  for *all* $i \in \{1, \dots, N\}$ **do**
    $D_i^{(t+1)} \leftarrow D_i^{(t)} \exp(-\alpha_t y_i h_t(\mathbf{x}_i))$
  end
  Normalize $D^{(t+1)}$ such that $\sum_{i=1}^{N} D_i^{(t+1)} = 1$
end
Output: $H(\mathbf{x}) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x})\right)$
return $H(\mathbf{x})$