Iraqi Journal of Computers, Communications, Control and Systems Engineering
Journal homepage: https://ijccce.uotechnology.edu.iq

Group-Based Sample Partitioning kNN: A Computationally Efficient kNN Algorithm for Resource-Constrained Environments

Ayad M Dalloo[1,*], Amjad J. Humaidi[2]

[1]College of Communication Engineering, University of Technology, Baghdad, Iraq
[2]College of Control and System Engineering, University of Technology, Baghdad, Iraq
[1]ayad.m.waiss@uotechnology.edu.iq, [2]amjad.j.humaidi@uotechnology.edu.iq

DOI: https://doi.org/10.33103/uot.ijccce.25.2.2

HIGHLIGHTS

- **Efficient Computational Reduction**: The proposed Group-Based Sample Partitioning (k_g^1 −kNN) Algorithm reduces distance computations usage to 21.79% of exhaustive kNN without any extra storage.
- **High Classification Performance:** Experimental evaluation on the Breast Cancer Dataset (WBC) demonstrates that k_g^1 −kNN maintains high classification accuracy (95.78%)
- **Scalability and Future Prospects:** The k_g^1 −kNN approach shows a 75.5% reduction in computation cost for larger datasets (100 to 30,000 samples) and indicates strong potential for scalable deployment in embedded systems.

ARTICLE HISTORY

Received: 03/ March /2025

Revised: 31/ March /2025

Accepted: 09/ May /2025

Available online: 30/ October /2025

ABSTRACT

The k-nearest neighbors (kNN) algorithm is widely adopted for classification due to its simplicity and effectiveness. However, its computational cost remains a significant challenge, particularly for resource-constrained environments with limited processing power and memory. This issue is addressed by proposing the Group-Based Sample Partitioning (k_g^1 −kNN) Algorithm, which introduces a two-phase approach to reduce computational complexity while maintaining classification accuracy. In the first phase, the algorithm pre-groups training samples by iteratively selecting anchor points and partitioning their k-nearest neighbors, thereby reducing redundancy in the dataset. In the second phase, the test sample dynamically selects local anchor points, constructing a smaller, more relevant neighborhood for efficient classification. Experimental results using the Breast Cancer Dataset from the UCI repository *(WBC) demonstrate that k_g^1 −kNN significantly reduces training and testing iterations while preserving high classification accuracy (95.78%), with a recall of 100%. Compared to exhaustive kNN, our approach achieves a substantial*

Keywords:
k-nearest neighbors, Data-level Approximate Computing, Approximate , Embedded System

reduction in distance computations approximately 78% lower without requiring additional memory. While the algorithm was tested on a relatively small dataset, k_g^1 −kNN shows promise for scalable implementation in embedded systems. Also, the proposed approach shows the computational cost can be reduced by over 75% for larger datasets when different datasets ranging from 100 to 30,000 samples were tested. This work specifically targets tabular datasets suitable for resource-constrained embedded systems. Therefore, the comparison primarily emphasizes exhaustive kNN, which serves as a clear baseline for computational complexity evaluation. Nevertheless, we have also provided comparisons with related works, highlighting methodological distinctions and similarities explicitly. Future work will explore an extended k_g^n −kNN framework, introducing multiple k-parameters for adaptive scaling to high-dimensional datasets while maintaining computational efficiency. https://github.com/AyadMDalloo/kg-kNN.

I. INTRODUCTION

The current trend in high-performance Very Large-Scale Integration (VLSI) design is increasingly centered on real-time digital signal processing (DSP) and machine learning algorithms. Such emphasis is critical in applications such as surveillance systems and wearable electronics, which necessitate the analysis and evaluation of sensed data to recognize patterns. In the context of the Internet of Things (IoT) and edge processing, quick action based on sensed data is imperative [1]. Many applications emphasize local processing over cloud computing to mitigate latency and connection limitations. However, local processing necessitates solutions that are energy-efficient, highly accurate, latency-sensitive, and cost-effective [2], [3]. The kNN algorithm has become more popular in machine learning applications due to its intuitive simplicity, strong statistical properties, robustness, and high accuracy across diverse applications. However, the implementation of the traditional kNN algorithm suffers from inefficiency and scalability issues, particularly with large datasets. This approach demands high computations, power consumption, and hardware resources because it compares each data point with all the other data points in the dataset [4].

Ongoing research has led to several proposed modifications to handle these shortcomings and improve the kNN algorithm's efficiency, accuracy, and scalability. For example, fast similarity search (FSS) methods are one of the kNN families that aim to accelerate similarity searches by organizing the data in structures that reduce unnecessary comparisons. For instance, there are K-Dimensional Tree (KD-Tree) [5] and Ball-Tree [6] algorithms in this family. These methods can classify datasets into hierarchical tree-based structures for faster retrieval of the closest neighbors, and this demands extra memory and computation costs. While KD-Tree performs well in low-dimensional datasets, their efficiency drops in high-dimensional data. Ball-Tree offers some improvements but still struggles in very high-dimensional datasets. These methods enhance kNN search efficiency but often face limitations in high-dimensional spaces. Approximate Similarity Search (ASS) methods are another family that focus on achieving fast retrieval by approximating nearest neighbors rather than computing exact distances. Therefore, this family is suitable for large-scale datasets. Locality Sensitive Hashing (LSH) [7] used randomized hash functions to map similar data points to the same bucket, enabling rapid

lookup. Spectral Hashing (SH) [8] improves upon LSH by learning data-specific hash functions, enhancing accuracy while maintaining efficiency. Product quantization (PQ) [9] further accelerates search by compressing feature vectors, reducing storage and computational costs in large-scale retrieval tasks. ckNN+, a clustering-based kNN method introduced by Gallego et al. [10], enhances efficiency by restricting searches to predefined clusters but lacks adaptive k-values and cluster augmentation. caKD+ kNN [10], an improved version of ckNN+ [11], addresses these limitations by incorporating adaptive k-values, cluster augmentation, and precomputed KD-Trees, significantly optimizing both efficiency and classification accuracy. However, caKD+ kNN has limitations due to higher preprocessing costs represented by clustering and KD-Tree construction, sensitivity to cluster quality, and additional hyperparameter tuning requirements for peak efficiency. Therefore, this caKD+ kNN is more complex than traditional kNN methods. The K1K2NN model [12] introduces a novel dual-stage neighbor-based approach for predicting multi-label side effects of COVID-19 drugs using chemical properties. This hybrid mechanism effectively combines drug-drug and side effect-side effect relationships, leading to high accuracy (up to 97.5%) on small, curated datasets. K1K2NN prioritizes prediction performance but incurs a higher computational cost due to pairwise distance and co-occurrence calculations across both samples and labels. Another work, the K²NN framework [13], proposes a pixel-level self-supervised learning method for remote sensing, using a hierarchical nearest neighbor approach to learn rich representations across image patches. It achieves notable efficacy on tasks like land cover classification and change detection, especially with limited labeled data. Main benefits include robust feature learning and efficient memory updates via submodular sampling. However, the method is computationally intensive. Although previous enhancements have improved computational efficiency, they often face limitations in storage overhead and high-dimensional data performance. This motivates the development of an efficient yet accurate variant suitable for constrained environments.

In this work, we propose the Group-based Sample Partitioning (k_g^1 −kNN) Algorithm, which tackles the high computational overhead of classical kNN by pre-grouping training data around "anchor" samples. During training, this design excludes each group's samples from the next to reduce the number of distance computations. During classification, k_g^1 −kNN finds the k nearest distances between "anchor" samples from fixed local groups and the test point. This operation helps to reduce computation and enhance classification accuracy without needing extra memories. The results show that k_g^1 −kNN is considered a lightweight and efficient extension to the kNN family that is particularly well-suited for resource-constrained environments.

The remainder of this paper is organized as follows: Section I discusses the fundamental principle of the standard (exhaustive) kNN. Section II describes the proposed methodology of the kNN model represented by the group-based sample partitioning ($k_g^1 - kNN$) algorithm. Section III presents the results of our experiments and discusses these findings in the context of existing research. Finally, Section IV concludes the paper with a summary of our findings, identifies the limitations, and discusses the expected solution for future work.

II. BACKGROUND

The k-Nearest Neighbors (kNN) algorithm is a simple, powerful method widely used for classification and regression tasks in machine learning. Conventional kNN is a non-parametric and lazy learning algorithm, meaning it makes no assumptions about the underlying data distribution and does not train a model in the traditional sense. It classifies new data points based on a majority vote from the k nearest samples, typically determined using Euclidean distance, as shown in *Fig. 1*. Even in its simplicity, kNN suffers from several challenges, including high computational cost, high dimensionality, and sensitivity to imbalanced data. The number of neighbors (k), distance metric choice, and feature scaling significantly influence its performance. The choice of k, the number of

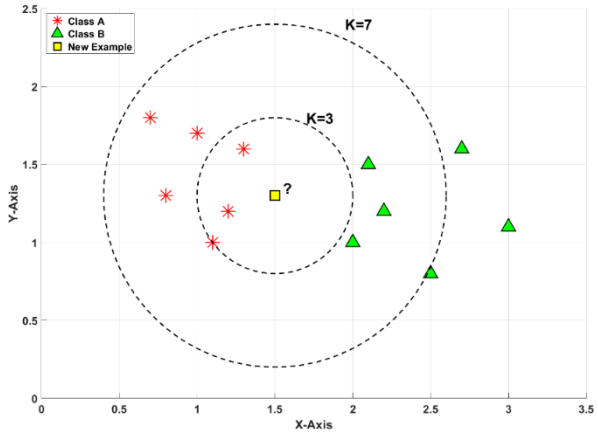

FIG. 1. ILLUSTRATION OF K-NEAREST NEIGHBORS (KNN) CLASSIFICATION WITH K=3 AND K=7.

neighbors, is crucial for the algorithm's performance, with a small k leading to potential overfitting and sensitivity to noise, while a large k may lead to underfitting. To find the best value for k, we need to iteratively test the dataset with cross-validation. However, optimizations like dimensionality reduction and efficient data structures (e.g., KD-Tree, Ball-Tree) can be used to speed up nearest neighbor search, and weighting techniques can improve its scalability and performance, making it effective in applications such as recommendation systems, anomaly detection, and text classification. Overall, kNN remains a popular choice in many real-world applications due to its flexibility and ease of implementation. However, we proposed a solution for designing efficient kNN architecture by introducing a dual-k approach, where the first k is used for data grouping or clustering, and the second k is employed to identify the k nearest neighbors for classification.

To identify the nearest neighbors, a distance metric is required. The most common distance metric used in kNN is Euclidean distance for continuous variables. Other distance metrics include:

1. Manhattan Distance: Used when features are more grid-like (e.g., city blocks).
2. Minkowski Distance: A generalized form of both Euclidean and Manhattan.
3. Cosine Similarity: Particularly useful when the magnitude of vectors (rather than distance) is important, like in text mining.

The k-nearest neighbors (kNN) algorithm can be mathematically represented through several steps for classification tasks: The k-nearest neighbors (kNN) algorithm operates by identifying the k closest points in a dataset to a given query point, based on a specified distance metric. The algorithm can be applied to both classification and regression problems, with predictions made based on the properties of the nearest neighbors. Let the training dataset $\mathcal{D}$ consist of n data points:

$$\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\} \tag{1}$$

where $(x_i \in R^d)$ represents a feature vector in d-dimensional space, and $y_i \in \{1,2,\dots,C\}$ for classification is the target label or value. For a query point $x_{\text{query}} \in R^d$, the goal is to predict the corresponding label $\hat{y}_{\text{query}}$. To determine the k-nearest neighbors, a distance metric $d(x_i, x_j)$ is used to compute the proximity between the query point x_{query} and each data point x_i in the training set. The Euclidean distance, commonly used in kNN, is defined in equation

$$d(x_{\text{query}}, x_i) = \sqrt{\sum_{k=1}^{d} (x_{\text{query},k} - x_{i,k})^2} \tag{2}$$

where $x_{\text{query},k}$ and $x_{i,k}$ represent the kth components of the query point and the training point, respectively. Other distance metrics, such as Manhattan distance, may also be employed depending on the application. Once the distances are computed between x_{query} and each x_i, the training points are

sorted by ascending distance. The set of the k-nearest neighbors, denoted as S_{kNN}, is formed from the closest k points:

$$S_{\text{kNN}} = \{(x_{i_1}, y_{i_1}), (x_{i_2}, y_{i_2}), \dots, (x_{i_k}, y_{i_k})\} \quad (3)$$

where $i_1, i_2, \dots, i_k$ represent the indices of the k nearest points in the sorted list. For classification tasks, the predicted label $\hat{y}_{\text{query}}$ is determined by majority voting among the k nearest neighbors. Formally, the predicted class is given by

$$y_{\text{query}} = \arg_y max \sum_{j=1}^{k} f(y_{ij} = y) \quad (4)$$

where $f(\cdot)$ is the indicator function, which returns 1 if the class of the jth neighbor ($y_{ij} = y$) and 0 otherwise. The class that appears most frequently among the neighbors is chosen as the predicted class. If distance-weighted voting is used, where closer neighbors have a higher influence, the prediction becomes:

$$\hat{y}_{\text{query}} = \arg_y \max \sum_{j=1}^{k} \frac{1}{d(x_{\text{query}}, x_{ij})} \cdot f(y_{ij} = y) \quad (5)$$

This method gives more weight to neighbors that are closer to the query point. In this case, neighbors that are closer to the query point contribute more remarkably to the predicted value. The k-nearest neighbors (kNN) algorithm has several hyperparameters and considerations that significantly impact its performance. One of the most critical hyperparameters is the number of nearest neighbors, denoted by *k*. The choice of *k* determines the balance between bias and variance: a small value of *k* makes the model sensitive to noise, increasing variance and potential overfitting, while a large *k* smooths predictions but may lead to underfitting by increasing bias. Moreover, the selection of an appropriate distance metric is critical to the success of kNN. Although Euclidean distance is frequently employed, it may not suit all datasets. Alternatives like Manhattan distance or cosine similarity might perform better in specific scenarios, such as data with grid-based layouts or high-dimensional sparse features, respectively. Equally important is normalizing features. Since kNN depends on distance calculations, features with varying scales can distort the model's outcomes when those with larger ranges may disproportionately sway results. Normalizing or scaling features to a uniform range ensures each attribute contributes equal weight in distance computations. To maximize the effectiveness of the kNN model for a specific task, these hyperparameters and preprocessing steps require careful adjustment.

III. METHODOLOGY

Before applying any algorithm, we normalize a dataset by implementing Min-Max Scaling, which is a common normalization technique used to scale data to a specific range, often between 0 and 1. The mathematical representation of this approach is straightforward and can be described as follows:

$$X_{\text{normalized},i,j} = \frac{X_{i,j} - \min(X_j)}{\max(X_j) - \min(X_j)}, \quad (6)$$

$$\forall i \in \{1, \dots, m\}, j \in \{1, \dots, n\}$$

Where $X \in R^{m \times n}$ is the original data matrix with m rows (data points) and n columns (features), $X_{\text{normalized}} \in R^{m \times n}$ is the normalized data matrix where each feature is scaled between 0 and 1. This process is applied independently to each feature (column) in the matrix.

After normalizing the dataset, we employed one of the data-level approximate techniques [1], [3] called the dynamic decision tree algorithm [1] for feature selection to identify key features and reduce computational cost. By selecting important features, the dimensionality fed into kNN is reduced, significantly decreasing computation, particularly for high-dimensional data. To address the sensitivity of decision trees and tendency to overfit, a repeated holdout cross-validation strategy (10 iterations) was applied and dynamically adjusted thresholds for feature selection. The dynamic decision tree algorithm

is summarized here, but the algorithms are discussed in detail in [1].

Algorithm II: Dynamic Decision Tree [1]

Inputs: Original features X, target labels y, number of iterations NoItr, and selection threshold thr.
Output: Final set of selected features.

Procedure:

A. Initialization:

- Initialize a feature counter S (zeros) matching the feature count in X.

B. Iterative Feature Selection: For NoItr iterations, repeat:

- Normalize/standardize dataset features.
- Split data into training and testing subsets (holdout cross-validation).
- Fit a decision tree on the training subset and compute feature importance.
- Rank features based on importance scores.
- Dynamically select features whose importance scores significantly exceed those of lower-ranked features.
- Update feature counter S with frequency of selected features.

C. Final Feature Selection:

- Identify features whose selection frequency exceeds the threshold (thr).
- Return these as the final selected features.

As discussed in Section II, traditional kNN's computational complexity motivates us to propose a dual-k approach (k_g^1 −kNN). The k_g^1 −kNN introduces a training stage employing a dual-k strategy to improve computational efficiency. The first k parameter (k_g) partitions and groups the training data into anchor-based subsets, reducing redundancy and computational cost. The second k identifies the nearest neighbors at classification time from these reduced subsets, effectively decreasing runtime complexity while preserving accuracy. This partitioning dramatically reduces redundant distance calculations, thus significantly decreasing computational complexity and runtime, especially critical for resource-constrained applications. The shortcoming of classical kNN is addressed here by an iterative selection mechanism: in the first phase, the algorithm repeatedly selects an anchor sample and partitions its k-nearest neighbors (the "group") from the training data, appending these groups to a reduced set while excluding them from further consideration. By removing redundant or densely clustered samples, we effectively shrink the training pool, thereby speeding up the later classification step. In the second phase, each test sample adaptively identifies several anchor points in the reduced training set; a local neighborhood is constructed for each anchor by taking a small window of samples, and then the nearest neighbors from this concatenated local subset are used to predict the test label via majority vote. Because classification thus depends on a smaller, more representative set of samples, the method substantially reduces the computational overhead at test time while still preserving diversity in the selected anchor points across the feature space. The mathematical formulation of the group-based sample partitioning kNN procedure is divided into two phases.

Algorithm $k_g^1 - kNN$

Inputs

1. $\mathbf{X_{train}} \in R^{N\times d}$:Training dataset (where N is the number of samples and d is the number of features).

2. $\mathbf{Y_{train}} \in R^{N\times \mathbb{1}} \in \{0,1\}$: Labels corresponding to the training dataset.

3. $\boldsymbol{X_{test}} \in R^{M\times d}$: Test dataset (where M is the number of test samples).

4. $\boldsymbol{k_g}$: Initial number of nearest neighbors to group and exclude at each iteration during the training-reduction phase.

5. **k**: Number of anchor points (in the partitioned set) used at test time to build the final neighborhood for classification.

Outputs

1. $\boldsymbol{predictions}$:A vector of size M containing the predicted labels for each test sample.

Phase 1: Group-Based Sample Partitioning with Sample Exclusion

Goal: Iteratively create *partitions* (or groups) of nearest neighbors from the training set and exclude them so that the final set is both smaller and more diverse. The final set (the reduced set) consists of only the anchor points.

1. **Initialization**

- Define the set of remaining indices: $\mathcal{R}^{(0)} = \{1,2,\dots,N\}$.
- Let selectedSamples=∅ and selectedLabels=∅.
- Set i←1.

2. **Iterative Selection**

While $\mathcal{R}^{(i-1)} \neq \emptyset$:

A- Anchor Selection

$$\boldsymbol{a_i} = \min \mathcal{R}^{(i-1)} \left(\text{for instance, take the first index in } \mathcal{R}^{(i-1)}\right)$$

$$\boldsymbol{x_{a_i}} \in \mathbf{X_{train}}(\boldsymbol{a_i}, :)$$

where $\boldsymbol{x_{a_i}}$ is the anchor sample.

B- Compute Distances

For each $j \in \mathcal{R}^{(i-1)}$, compute

$$d_{a_i}(j) = d\left(\boldsymbol{x_{a_i}}, \boldsymbol{x_j}\right).$$

C-Identify Nearest Neighbors: Find a set

$$\mathcal{N}_i = \arg \min_{\substack{S\subset\mathcal{R}^{(i-1)} \\ |S| = min\left(k_g, |\mathcal{R}^{(i-1)}|\right)}} \left\{d_{a_i}(j): j \in S\right\}$$

In other words, $\mathcal{N}_i$ is the set of the closest $min\left(k_g, \left|\mathcal{R}^{(i-1)}\right|\right)$ indices in $\mathcal{R}^{(i-1)}$.

D-Append Partition (Group) to Selected Set

$$\text{selectedSamples} \leftarrow \text{selectedSamples} \cup \left\{\boldsymbol{x_j}: j \in \mathcal{N}_i\right\},$$

$$\text{selectedLabels} \leftarrow \text{selectedLabels} \cup \left\{y_j: j \in \mathcal{N}_i\right\}.$$

E-Sample Exclusion: Remove $\mathcal{N}_i$ from $\mathcal{R}^{(i-1)}$ for the next training iteration

$$\mathcal{R}^{(i)} = \mathcal{R}^{(i-1)} \setminus \mathcal{N}_i.$$

F-Increment

$$i \leftarrow i + 1.$$

(Optionally, adjust $\boldsymbol{k_g}$as needed; in this algorithm $\boldsymbol{k_g}$ remains the same.)

3. **Result of Phase 1:** After the loop completes (when $\mathcal{R}^{(i)} = \emptyset$), we have the partitioned training dataset and their anchor samples will be used in the second phase (Testing Phase)

$$\boldsymbol{X_{partitioned}} = \text{selectedSamples}, \boldsymbol{Y_{partitioned}} = \text{selectedLabels}.$$

These form the final reduced (partitioned) training set.

Phase 2: Adaptive Neighborhood Testing

Goal: For each test sample $\boldsymbol{x_t^{(t)}}$, adaptively select anchor points from the partitioned training set and build a local neighborhood for classification.

1. **Initialization**
   - Let $L = |X_{partitioned}|$
   - Define an index set
   $$A = \{1,1 + k_g, 1 + 2\,k_g, \dots\} \cap \{1,2, \dots, L\},$$
   which serves as **anchor indices** in the reduced set.
2. **Anchor Point Selection:** For a single test sample $x_t^{(t)}$, compute the distance to each anchor index a ∈*A*:
   $$d_t(a) = d(x_t^{(t)}, \boldsymbol{X}_{partitioned}[a])$$
   Sort these anchor distances in ascending order, and denote the top-k anchors as:
   $$\mathcal{M}_t = arg \min_{\substack{S \subset A \\ |S|=k}} \{d_t(a): a \in S\}.$$
3. **Local Group Construction:** For each anchor $m \in \mathcal{M}_t$, define a local group:
   $$\mathcal{G}(m) = \{m, m + 1, \dots, m + (k_g - 1)\} \cap \{1,2, \dots, L\}.$$
   Concatenate (union) these groups to form:
   $$\mathcal{S}_t = \bigcup_{m \in \mathcal{M}_t} \mathcal{G}(m).$$

The set $\mathcal{S}_t$ collects the local samples from the reduced training data around each chosen anchor.

4. **Distance Computation in the Local Set:** For each $s \in \mathcal{S}_t$:
   $$d_t'(s) = d(x_t^{(t)}, \boldsymbol{X}_{partitioned}[s])$$
   Sort these by ascending order:
   $$\mathcal{S}_t' = arg \min_{\substack{Q \in \mathcal{S}_t \\ |Q|=k}} \{d_t'(s): s \in Q\}$$
   $\mathcal{S}_t'$ denotes the top-*k* nearest samples from the local group.

5. **Majority Voting (or Other Rule)**

   Let $\boldsymbol{Y}_{partitioned}\,[s]$ be the label corresponding to sample *s* in $X_{partitioned}$. The label prediction for $x_t^{(t)}$is given by:
   $$\hat{y}_t = \text{mode}\{\boldsymbol{Y}_{partitioned}\,[s] : s \in \mathcal{S}_t'),$$
   i.e., the majority vote among the *k* nearest samples in the local set.
6. **Complete Predictions:** Repeat Steps (2)–(5) for each test sample *t*=1,2,…,*M*. Collect all predicted labels into:
   $$\boldsymbol{Predictions} = [\,\hat{y}_1, \hat{y}_2, \dots, \hat{y}_M\,]^\top$$

The classical kNN algorithm does not require a formal "training" phase; consequently, it uses the entire dataset at testing time, which can significantly increase the computational cost for large datasets.

In contrast, our proposed approach introduces a training stage to reduce the dataset size and thereby improve efficiency. To analyze the number of iterations or the computational cost in both training and testing phases, let N represent the total number of training samples and k_g denote the number of samples per group. The total number of complete groups that can be formed is defined as:

$$M_g = \left\lfloor \frac{N}{k_g} \right\rfloor \tag{7}$$

where $\lfloor \cdot \rfloor$ represents the floor function. The remainder of the samples that do not form a complete group is expressed as $\mathrm{mod}(N, k_g)$, where mod(·) denotes the modulus operation. The summation function Tot_{tr}, which evaluates a metric over all groups and the remainder, is given by

$$Tot_{tr}(k_g) = \sum_{n=0}^{M_g} |N - k_g \cdot n - 1| + \mathrm{mod}(N, k_g) \tag{8}$$

To evaluate computational cost during the testing phase of a nearest-neighbor-based algorithm (e.g., k-nearest neighbors or a similar grouping-based method), the total computation cost, denoted as Tot_{test} , can be expressed as:

$$Tot_{test}(k_g) = M_g + \mathrm{M}^* k_g * k \tag{9}$$

The formula breaks down the total computation cost into two primary components: Group-Level Cost (M_g) and Sample-Level Cost ($\mathrm{M} * k_g * k$). Where M represents the total number of testing samples, representing the instances for which predictions or classifications are to be made. M_g represents the number of complete groups within the dataset, indicating the organizational structure of the training samples (e.g., groups or clusters that partition the dataset). It represents a fixed cost that is independent of the number of testing samples (M) k_g denotes the number of samples per group, representing the size of each group that is processed during the testing phase. This can be interpreted as the number of training samples accessed from each group during distance or similarity computations. k denotes the number of candidate neighbors considered for each testing sample, which influences the precision of the nearest neighbor search. We combine both equations 8 and 9 to get:

$$\mathrm{Tot}(k_g) = \mathrm{Tot}_{\mathrm{tr}}(k_g) + \mathrm{Tot}_{\mathrm{test}}(k_g) \tag{10}$$

Given fixed (k, N, M), the problem is to find optimal k_g

$$k_g^* = \underset{k_g \in \{1, \ldots, N\}}{arg\,min} \; Tot_{test}(k_g) \tag{11}$$

One of the primary benefits of this approach lies in its efficiency, as removing redundant samples accelerates classification, especially in large datasets. Moreover, focusing on "key" anchor samples can yield better generalization by providing a more balanced spread across different classes and mitigating overfitting in dense clusters. Although the iterative selection can be large, as shown in the equation below. In the worst case, the resulting subset is generally smaller, thus improving scalability and reducing test-time overhead. Finally, the method's robustness stems from adaptive anchor selection at test time, which accounts for local variations in data density and is especially advantageous in high-dimensional spaces.

IV. EXPERIMENTAL EVALUATION AND PERFORMANCE ANALYSIS

In the previous section, we proposed a training phase for classical kNN to reduce the computation costs of its classification phase. In this section, we assess our proposed k_g^1 −kNN algorithm to find the optimal value of the grouping neighbors constant k_g, which demonstrates improved computational efficiency while maintaining high classification performance. We introduced a two-phase strategy of kNN: in the first phase, we partition the sample set and pre-group neighbors; the algorithm addresses the computational challenges of classical kNN, particularly for large datasets. To validate this proposed approach, we used the Breast Cancer Dataset from the UCI repository (WBC) [14]. Before applying the k_g^1 −kNN algorithm, we normalize and select the important features using the dynamic decision tree algorithm [1]**,** as discussed in the previous sections. The WBC dataset was reduced to 286 samples × 3 features. Feature selection was applied to the Wisconsin Breast Cancer (WBC) dataset to remove redundant and less informative features, thereby enhancing computational efficiency and reducing model complexity. The three most discriminative features were selected based on a dynamic decision tree-based feature importance analysis, considering their relevance, statistical significance, and predictive power. Without identifying which features significantly contribute to accurate diagnosis, incorrect model predictions might inadvertently be considered valid during validation and training phases, potentially leading to serious diagnostic errors in medical applications. To mitigate this risk, we conducted three rigorous evaluations to assess the impact of feature and instance reductions: (1) tenfold cross-validation, (2) evaluation solely on the training subset, and (3) evaluation using the complete dataset. These analyses confirmed that the dataset reduction did not degrade accuracy, demonstrating that the proposed model maintains reliability even after substantial feature and instance selection. We needed to assess our proposed model to make sure it is a robust model and characterize the model by generalization. Consequently, a 10-fold cross-validation strategy was employed to construct a training set of 258 samples and a testing set of 28 samples in each fold. Cross-validation showed stable performance across folds after feature selection, suggesting improved generalization due to the elimination of noisy and redundant features.The evaluation considers multiple metrics, including the number of iterations for training and testing, delay, accuracy, precision, recall, and F1-score, to assess the algorithm's scalability and adaptability. All models were implemented on a desktop computer running MATLAB R2024a. With the code and datasets, we provide online and the rest of the results obtainable, enabling other researchers to utilize the prepared datasets with their features and labels already separated. Table IV in Appendix, presents the various experimental arrangements made and the important packages needed to run the prediction models and to evaluate the techniques.

Table I demonstrates the impact of feature selection using a Dynamic Decision Tree (DDT) on the performance and efficiency of the k_g^1 −kNN model applied to the WBC dataset. Reducing the number of features from 9 to 3 leads to a 1.8% improvement in accuracy (from 94.1% to 95.8%) and a 4.0%

TABLE I. IMPACT OF FEATURE SELECTION ON THE PERFORMANCE OF THE k_g^1 −KNN MODEL

Model	Feature Selection Approach	Datasets	Binary Classification Metrics				Saving % [M+T+P]
			Accuracy	Precision	Recall	F1	
k_g^1 −kNN (k_g = 9)	N/A	WBC [9 FEs]	94.1	96.0	96.2	96.0	N/A
	DDT	WBC [3 FEs][1]	95.8	94.4	100	97.0	M: 67% P: 67% T=67% Speedup: 1%

[1] [4 3 9], M: Memory, T: Training, P: Predicting

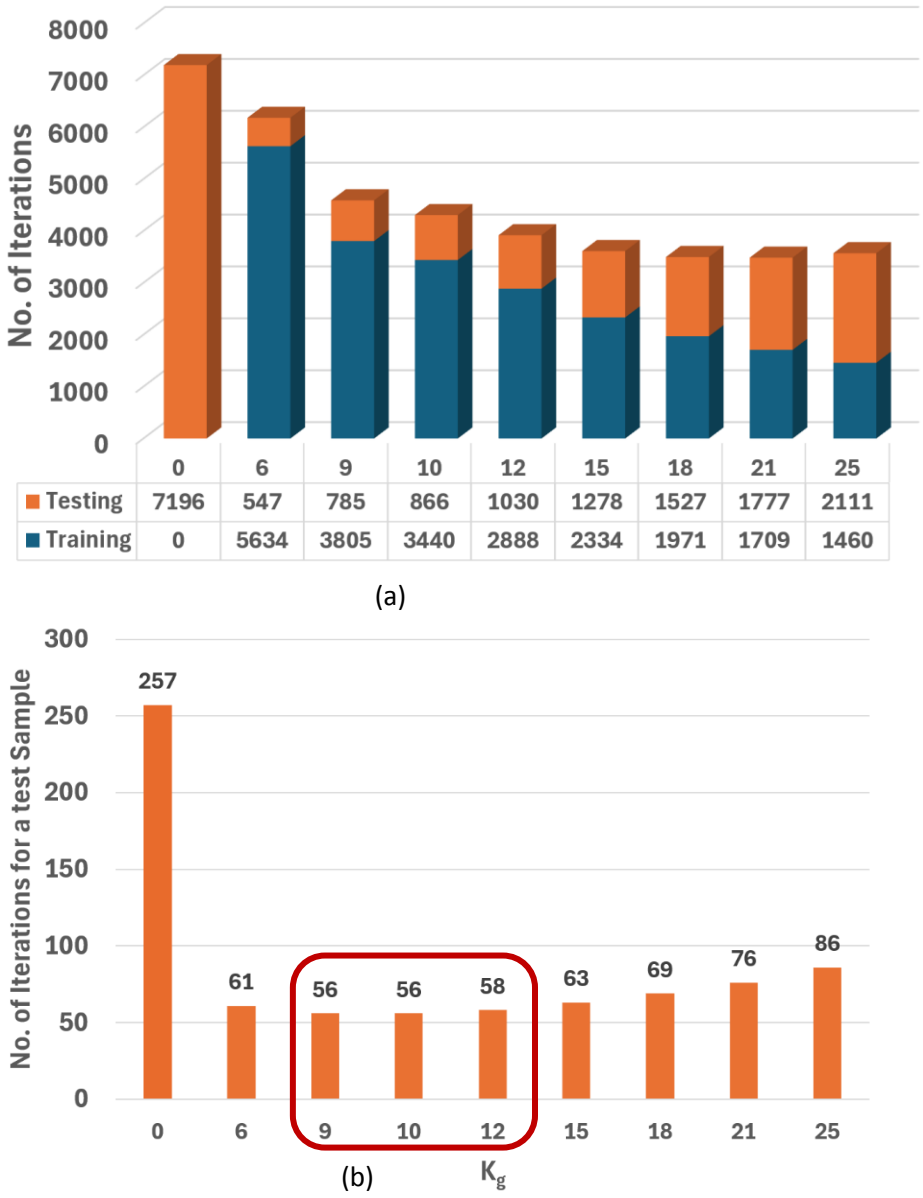

FIG.2. PERFORMANCE EVALUATION OF $k_g^1 - kNN$:(A) TRAINING AND TESTING ITERATIONS VS. k_g, (B) SINGLE-SAMPLE TESTING ITERATIONS.

FIG.3. DELAY OBSERVED DURING TESTING PHASE AS A FUNCTION OF K_G.

improvement in recall (from 96.2% to 100%), while the F1-score increases by 1.0% (from 96.0% to 97.0%). Although precision slightly drops by 1.7% (from 96.0% to 94.4%), the overall classification performance remains robust. Importantly, the reduced feature model achieves 67% savings across memory, training, and prediction computations, highlighting significant efficiency gains. The overall computational speedup, though modest at 1%, reinforces the method's suitability for resource-constrained applications

The findings, supported by figures and data, reveal critical insights into the algorithm's behavior under varying parameter settings, specifically the number to rise again, which causes the computing cost in the testing phase, while our goal is to have the lowest number of nearest neighbors (k_g) used for pre-grouping. The first test examines the number of iterations required during the training and testing phases as k_g increases, as shown in *Fig. 2a*. Without pre-grouping (k_g=0), there is no training phase, and the testing phase requires the entire training dataset, which results in an unmanageable computational cost. This inefficiency is reflected in both training and testing iterations, which are excessively high compared to other k_g values. However, as k_g increases, the pre-grouping mechanism significantly reduces the size of the training set, thereby minimizing redundant computations. The number of iterations stabilizes in the range of k_g = 9 to k_g = 12, marking an optimal balance where both training and testing phases achieve computational efficiency without losing accuracy. Beyond this range (k_g > 12), testing iterations begin the testing phase with the lowest possible training phase. The next test facilitates identification of the optimal k_g.

Fig. 2b illustrates the number of iterations required during the testing phase for a single test sample as a function of k_g, the number of nearest neighbors used during pre-grouping. Without grouping (k_g=0), the algorithm requires 257 iterations, as the entire training dataset is used for comparison. For moderate values of k_g (e.g., k_g = 9, 10, 12), the number of iterations stabilizes between 56 and 58, marking an optimal region where computational costs are reduced without sacrificing accuracy. Beyond k_g = 15,

TABLE II. PERFORMANCE METRICS OF $\boldsymbol{k_g^1 - kNN}$ FOR VARIOUS VALUES OF k_g

k_g	6	9	12	18	21	25
Delay	0.03502	0.03018	0.03148	0.03828	0.03988	0.04137
Accuracy	95.825	95.776	95.813	95.813	95.813	95.788
Precision	94.583	94.437	94.374	94.467	94.224	94.386
Recall	100	100	100	100	100	100
F1_Score	97.183	97.040	97.085	97.069	96.936	97.066

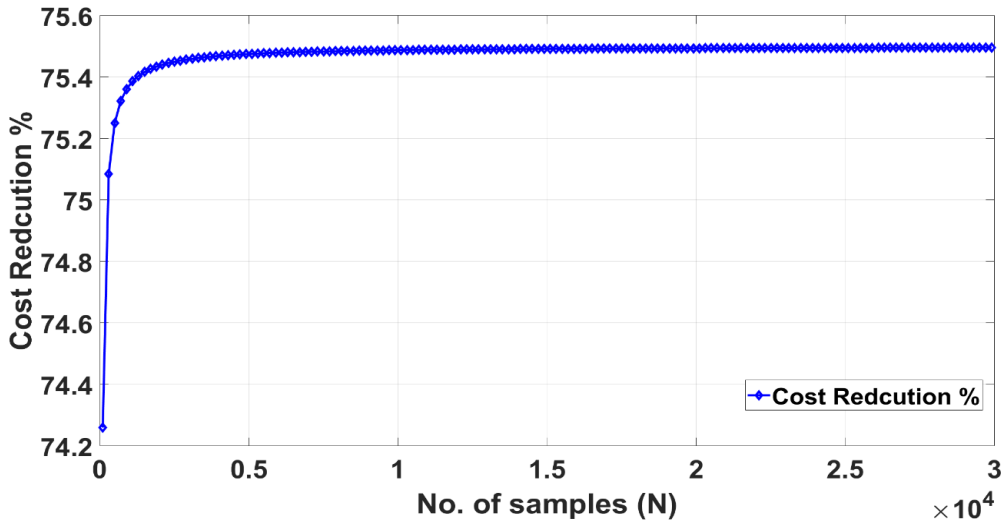

FIG.4. COMPUTATIONAL EFFICIENCY OF THE PROPOSED $k_g^1 - kNN$ ALGORITHM IN REDUCING COMPUTATION AS THE DATASET SIZE INCREASES.

testing costs gradually rise again due to overly coarse grouping, which diminishes dataset granularity and forces the algorithm to process more samples during classification. This highlights the importance of selecting an appropriate k_g to balance efficiency and precision. After finding the optimal region of k_g, we want to show how this modification of kNN will affect computation and delay.

Delay, representing the computational latency during the testing phase, exhibits a similar trend. For smaller values of k_g (e.g., k_g = 6, 9, 12), delay decreases, with the lowest observed delay at k_g = 9 (0.03018), as shown in *Fig. 3* and Table II. This reduction occurs because moderate grouping reduces the dataset size while retaining representative samples, allowing efficient neighborhood construction. However, as k_g increases beyond 12, delay begins to rise, peaking at k_g = 25 (0.04137 sec). This increase is caused by the loss of dataset granularity at higher k_g values, forcing the algorithm to process a larger number of samples during neighborhood construction.

These results emphasize the importance of selecting an optimal k_g value to minimize delay while preserving classification performance. The classification performance of k_g^1 −kNN was assessed using accuracy, precision, recall, and F1-score, which remain consistently high across all k_g values. The accuracy is within a range of 95.78% to 95.83%. This indicates that the algorithm's ability to correctly classify samples is largely unaffected by changes in k_g, as shown in Table II. Recall, which quantifies the algorithm's ability to identify all relevant samples, remains consistently at 100% across all k_g values. This stability demonstrates the algorithm's robustness in minimizing false negatives, making it highly reliable for critical applications where recall is prioritized. However, we compared our proposed approach with other modified kNN algorithms.

In the second test, we find the optimal k_g for different datasets ranging from 100 to 30,000 samples, using equation 10. In the current study, the optimal values for the grouping parameter k_g were determined via an iterative search process. Specifically, we varied k_g across a predefined range and evaluated its impact on the number of distance computations during the testing phase. The selection criterion was based on identifying the value (or range) of k_g that minimized the number of distance calculations while maintaining classification accuracy. By using optimal k_g, we could find the minimum value of the total computation cost, which helps us to calculate the percentage cost reduction achieved by the proposed method with respect to the number of samples N, as shown in *Fig. 4*. This metric, defined as equation 12, indicates how effectively the approach reduces the computational overhead compared to an exhaustive kNN. Early on (at smaller dataset N), the cost reduction hovers around 74.2%; then it rises sharply and stabilizes at about 75.5% for larger datasets. This leveling-off

effect suggests that as the dataset grows, the relative efficiency of the method remains high, consistently preserving over 75% savings in computational cost. Then we can calculate

$$\text{Percentage Cost Reduction} = \frac{1}{N_{itr}}\sum_{i=1}^{N_{itr}} \frac{Total\ Computation\ Cost\ of\ \text{k}^2\text{NN}}{Total\ Computation\ Cost\ of\ exhaustive\ \text{kNN}} \times 100\% = \frac{1}{N_{itr}}\sum_{i=1}^{N_{itr}} \frac{Tot_i}{N^2} \times 100\% \quad (12)$$

Where N_{itr} is the number of tested sample datasets.

TABLE III. COMPARISON OF $\boldsymbol{k_g^1 - kNN}$ WITH OTHER MODIFIED KNN ALGORITHMS, INCLUDING EXHAUSTIVE KNN, CAKD+, AND KD-TREE.

Features	Exhaustive kNN	caKD+[10] 2022	KD-Tree[15] 2018	k_g^1 −kNN (proposed)
Dimension Reduction	N/A	Deep Neural network	sparse reconstruction model+ locality-preserving regularization	**Approximate Data-level techniques (Feature Selection)**
Datasets	WBC	**Different**	**Different**	WBC
Grouping and clustering	N/A	k-means	K*tree	**K grouping and excluding**
training complexity	$O(N^2)$[1]	The complexity of DNN+ Clustering+ adaptive k+ Cluster Augmentation+ KD-Tree Construction[2] $O(N^{1.5}\log N) + O(E \cdot L \cdot N \cdot M^2)$	$O(N*\log_2(N))$	$\boldsymbol{O(\sum_{n=0}^{M_g} \lvert N - k_g \cdot n - 1\rvert + \mathbf{mod}(N, k_g)) \approx O(\frac{N^2}{2 \cdot k_g})}$
Classification Accuracy	96.94	**96.31**	94.71	N/A
	95.8251	N/A	N/A	**95.7882**
Efficiency (Distance Calculations %)	100.00	**0.19**	28.41	24.529
Advantages	Simple and easy to implement.	Near-optimal classification accuracy with minimal distance computations (only 0.19% of exhaustive kNN).	Effective for low-dimensional data.	**Simple and low cost. Maintain the accuracy high**
Limitations	Computationally expensive (requires all pairwise distance computations).	Relatively complex pipeline (requires clustering, feature extraction, and adaptive parameters).	**Performance degrades significantly in high-dimensional datasets ("curse of dimensionality").**	**Testing only on Tabular datasets**

[1] where n is the number of training samples, [2]L: number of layers, M: number of neurons per layer, N: number of the training samples, E: number of epochs

The evaluation consistently identifies the range k_g = 9 to k_g = 12 as the optimal range for k_g^1 −kNN performance. In this range, the algorithm achieves the best trade-off between computational efficiency, delay, and classification accuracy. In this work, we showed the performance evaluation of the proposed k_g^1 −kNN algorithm for small datasets like the breast cancer dataset, which considered a limitation of this work. But the results highlight the scalability and adaptability of the proposed algorithm, especially for large datasets where classical kNN struggles with computational overhead. We will discuss the

limitations and future work in the next section. Overall, k_g^1 −kNN proves to be a powerful alternative to classical kNN by offering a scalable solution for real-world applications. Table III compares k_g^1 −kNN with other kNN variants, including exhaustive kNN, caKD+ [10], and KD-Tree [15], based on key features such as dimension reduction, dataset usage, grouping methods, training complexity, classification accuracy, and efficiency. The proposed k_g^1 −kNN approach leverages approximate data-level techniques to improve computational efficiency while maintaining high classification accuracy (95.78%). Unlike traditional kNN, which has a high computational cost, k_g^1 −kNN significantly reduces distance calculations by 78.29% of exhaustive kNN. However, it requires further testing on high-dimensional datasets, where additional k parameters might enhance performance without extra storage overhead.

V. CONCLUSION, LIMITATIONS, FUTURE DIRECTIONS

The proposed k_g^1 −kNN algorithm effectively addresses the computational inefficiencies of classical kNN by integrating approximate data and architecture levels, techniques are represented by feature selection algorithms and group-based sample partitioning (a dual-k strategy), respectively. Our evaluation, conducted on the Breast Cancer Dataset from the UCI repository (WBC), demonstrates that k_g^1 −kNN significantly reduces computational costs while maintaining competitive classification accuracy. Experimental results validate the method's scalability and efficiency, notably reducing distance computations by about 78% compared to exhaustive kNN. This highlights k_g^1 −kNN's strong potential for real-world applications in computationally constrained environments. However, the findings also reveal that the efficiency of k_g^1 −kNN relies on proper parameter selection, particularly k_g (grouping).

One of the primary limitations of k_g^1 −kNN is that it has been tested only on a relatively small dataset (WBC). Although we tested the proposed algorithm to show the computational cost saving increases with dataset size, achieving approximately 75.5% reduction for datasets ranging from 100 to 30,000 samples. While the algorithm performs efficiently in this setting, its scalability to high-dimensional datasets remains an open question. Additionally, the current approach uses only two k parameters—one for grouping (k_g) and another for classification (k). This limitation could impact its adaptability to more complex datasets where a more dynamic approach might be needed. Furthermore, while the algorithm reduces computational overhead compared to exhaustive kNN, it still requires more testing computations than advanced indexing-based approaches (e.g., KD-Tree, locality-sensitive hashing). Finally, while the approximate data-level techniques reduce redundancy in the training set, additional approximate components could further improve efficiency.

TABLE IV. DETAILS OF THE EXPERIMENTAL SETUP AND SOFTWARE ENVIRONMENT

No.	Name	Version
1	Operating System	Windows 10 Pro, 22H2, 64-bit
2	Processor	Intel(R) Core (TM) i7-7700 CPU @ 3.60GHz, x64-based processor
3	RAM	16 GB
4	Hard disk	2TB SSD and 1TB
5	MATLAB	2024a, 64-bit
6	MS Excel	Professional Plus 2024

For future work, Future work aims to extend the proposed k_g^1 −kNN algorithm effectively to high-dimensional datasets. Preliminary experiments have already been conducted using hierarchical multi-level partitioning, adaptive selection of multiple grouping parameters (k_g), and incorporating dimensionality reduction techniques such as Principal Component Analysis (PCA) and autoencoders. These initial results show promise in managing the increased complexity without substantially raising computational overhead or memory usage. To extend k_g^1 −kNN to high-dimensional datasets, we propose introducing additional k_g parameters instead of using just two (k_g and k). A hierarchical or tree-like approach could be explored without requiring extra storage, as we can simply store multiple k parameters in registers rather than maintaining large indexing structures. This enables the algorithm to scale properly while maintaining its lightweight storage advantage. Moreover, we can integrate the other approximate computing techniques into k_g^1 −kNN to further accelerate classification. To truly assess the effectiveness of this algorithm, the k_g^1 −kNN algorithm also requires additional testing on larger and more heterogeneous datasets composed of high-dimensional feature spaces. Future investigations will further evaluate and refine these methodologies, particularly exploring how hierarchical approaches combined with adaptive dimensionality reduction can optimize performance and scalability in high-dimensional feature spaces.

CONFLICT OF INTEREST

There is no conflict of interest in this research.

DATA AVAILABILITY STATEMENT AND SUPPLEMENTARY MATERIALS

The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author. The source code, further results and Datasets in extension ".mat" are publicly available online https://github.com/AyadMDalloo/kg-kNN.

ACKNOWLEDGMENT

I sincerely thank Ms. Sulaf M. Waiss for meticulously reviewing my paper and providing valuable feedback.

REFERENCES

[1] A. M. Dalloo and A. J. Humaidi, "Optimizing Machine Learning Models with Data-level Approximate Computing: The Role of Diverse Sampling, Precision Scaling, Quantization and Feature Selection Strategies," *Results in Engineering*, vol. 24, p. 103451, Dec. 2024, doi: 10.1016/j.rineng.2024.103451.

[2] A. M. Dalloo, A. J. Humaidi, A. K. A. Mhdawi, and H. Al-Raweshidy, "Low-Power and Low-Latency Hardware Implementation of Approximate Hyperbolic and Exponential Functions for Embedded System Applications," *IEEE Access*, vol. 12, pp. 24151–24163, 2024, doi: 10.1109/ACCESS.2024.3364361.

[3] A. M. Dalloo, A. Jaleel Humaidi, A. K. Al Mhdawi, and H. Al-Raweshidy, "Approximate Computing: Concepts, Architectures, Challenges, Applications, and Future Directions," *IEEE Access*, vol. 12, pp. 146022–146088, 2024, doi: 10.1109/ACCESS.2024.3467375.

[4] H. S. Laxmisagar and M. C. Hanumantharaju, "FPGA implementation of breast cancer detection using SVM linear classifier," *Multimed Tools Appl*, vol. 82, no. 26, pp. 41105–41128, Nov. 2023, doi: 10.1007/s11042-023-15121-6.

[5] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An Algorithm for Finding Best Matches in Logarithmic Expected Time," *ACM Trans. Math. Softw.*, vol. 3, no. 3, pp. 209–226, Sep. 1977, doi: 10.1145/355744.355745.

[6] T. Liu, A. W. Moore, and A. Gray, "Efficient exact k-NN and nonparametric classification in high dimensions," in *Proceedings of the 17th International Conference on Neural Information Processing Systems*, in NIPS'03. Cambridge, MA, USA: MIT Press, Dec. 2003, pp. 265–272.

[7] M. Bawa, T. Condie, and P. Ganesan, "LSH forest: self-tuning indexes for similarity search," in *Proceedings of the 14th international conference on World Wide Web*, in WWW '05. New York, NY, USA: Association for Computing Machinery, May 2005, pp. 651–660. doi: 10.1145/1060745.1060840.

[8] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Proceedings of the 22nd International Conference on Neural Information Processing Systems*, in NIPS'08. Red Hook, NY, USA: Curran Associates Inc., Dec. 2008, pp. 1753–1760.

[9] H. Jegou, M. Douze, and C. Schmid, "Product Quantization for Nearest Neighbor Search," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, no. 1, pp. 117–128, Jan. 2011, doi: 10.1109/TPAMI.2010.57.

[10] A. J. Gallego, J. R. Rico-Juan, and J. J. Valero-Mas, "Efficient k-nearest neighbor search based on clustering and adaptive k values," *Pattern Recognition*, vol. 122, p. 108356, 2022, doi: https://doi.org/10.1016/j.patcog.2021.108356.

[11] A.-J. Gallego, J. Calvo-Zaragoza, J. J. Valero-Mas, and J. R. Rico-Juan, "Clustering-based *k*-nearest neighbor classification for large-scale data with neural codes representation," *Pattern Recognition*, vol. 74, pp. 531–543, Feb. 2018, doi: 10.1016/j.patcog.2017.09.038.

[12] P. Das and D. H. Mazumder, "K1K2NN: A novel multi-label classification approach based on neighbors for predicting COVID-19 drug side effects," *Comput. Biol. Chem.*, vol. 110, no. C, Jun. 2024, doi: 10.1016/j.compbiolchem.2024.108066.

[13] J. Yuan, Y. Xu, and Z. Wang, "K2NN: Self-Supervised Learning with Hierarchical Nearest Neighbors for Remote Sensing," in *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2023, pp. 1–5. doi: 10.1109/ICASSP49357.2023.10096425.

[14] Wi. Wolberg, "Breast Cancer Wisconsin (Original)." 1990.

[15] S. Zhang, X. Li, M. Zong, X. Zhu, and R. Wang, "Efficient kNN Classification With Different Numbers of Nearest Neighbors," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 5, pp. 1774–1785, May 2018, doi: 10.1109/TNNLS.2017.2673241.