Accelerating GridSearchCV Hyperparameter Tuning Method Using an FPGA-based Hardware Accelerator

Arian Shahbazian[1], Mohammadreza Binesh Marvasti[1*], Seyyed Amir Asghari[2]

[1]Department of Electrical and Computer Engineering, Kharazmi University, Tehran, Iran.

[2]Data Science and Technology Department, University of Tehran, Tehran,Iran.

*Corresponding author(s). E-mail(s): marvasti@khu.ac.ir; Contributing authors: arian.shahbazian@khu.ac.ir; s.a.asghari@ut.ac.ir;

Abstract

Machine learning models underpin critical applications, and their performance hinges on effective hyperparameter tuning , yet exhaustive methods like GridSearchCV are computationally intensive on general-purpose systems. This paper presents a custom FPGA accelerator for the GridSearchCV–KNN pipeline, simulated on an Intel Cyclone V platform. The six-module RTL design features an on-chip unique memory unit, a pipelined Manhattan-distance subtractor, and a parallel-sorting Distance Memory that tracks nearest neighbors without explicit sorting. A dedicated hardware controller manages hyperparameter grid enumeration and K-fold evaluation entirely in hardware. We introduce 'CV-full' (Leave-One-Out Cross-Validation) to test our system under the most intensive circumstances. Post-synthesis simulations show average speedups of 1.8×, 2×, and about 10× under 5-fold, 10-fold, and CV-full settings, respectively, with a peak of 27.7× over a Google Colab server. By minimizing off-chip memory access and explicit sorting stages, our FPGA design offers a special-purpose hardware solution for accelerating GridSearchCV–KNN hyperparameter tuning.

Keywords: FPGA, GridSearchCV, Hardware Accelerator, Hyperparameter Tuning, Machine Learning

1 Introduction

Artificial intelligence (AI) and machine learning (ML) are increasingly being applied across many domains [1], such as medical science, financial management, recommendation systems, and the internet of things. These fields have a significant impact on our daily lives. For example, ML algorithms such as K-Nearest Neighbor (KNN), Random Forest (RF), and Support Vector Machine (SVM) are widely used for classification and prediction of vital diseases [2], such as cancer and diabetes. Therefore, enhancing the accuracy and performance of these well-known ML algorithms could be highly beneficial. In terms of increasing the accuracy, ML algorithms can be tuned with some parameters. Nearly all ML algorithms include some parameters that have to pre-configured before the learning process starts. These parameters are called Hyperparameter (HP), and tuning them carefully will enhance the accuracy of ML algorithms [3]. For Example, HP 'N-Neighbors' or 'K' has a notable effect on the KNN algorithm accuracy [4]. Initially, the selection of HPs was a manual process, which proved to be highly time-consuming and error-prone due to the vast number of possible HPs to be checked. Later, HP auto-tuning methods changed everything; They decreased the time and errors of the HP selection process. GridSearchCV (GSCV), for instance, is now getting used widely as the most popular HP tuning method to find the best possible combination of different ML algorithms' HPs, and eventually it increases their accuracy [5]. Although HP auto-tuning methods are much faster than the manual process, they are still slow due to the large number of possible combinations to evaluate and the significant computational resources required. GSCV is a complete brute-force method, which examines and tests every possible combination of HPs on the dataset, and it takes a long time to complete tuning and selecting the best HPs [6]. In total, these methods are complex and time-consuming, particularly in case of working on high-dimensional datasets. In conclusion, HP tuning methods are very effective on ML algorithms' accuracy; however, they are suffering from being slow. Hence, finding a solution to accelerate HP tuning methods does really matter.

Nowadays, about 25 quintillion (10**18) bytes of data are being produced daily [7]. It means ML algorithms require more computational power than ever to effectively and efficiently convert enormous amounts of data into useful insights. Undoubtedly, simple and average computers are not capable enough to do such extreme computational tasks anymore. To solve this issue, hardware accelerators like GPU, TPU, ASIC, and Field Programmable Gate Array (FPGA) are widely used as co-processors in computationally intensive tasks [8]. They offer higher performance and productivity than general-purpose processors, and it has made them very popular among AI and ML developers. Studies on FPGAs have often demonstrated higher performance, lower power consumption, and reduced resource usage in specific applications when compared to general-purpose processors such as CPUs [9]. FPGAs are frequently recognized for their architectural flexibility, high computational throughput, energy efficiency, and reliability, which can make them well-suited for accelerating compute-intensive tasks such as neural networks [10], and potentially for GSCV. This paper presents a hardware accelerator architecture to accelerate the performance of the GSCV-KNN hyperparameter tuning method, outperforming its software-based counterpart. The proposed architecture is FPGA-based and has been implemented and simulated using the VHDL language. The proposed architecture will be compared with several general-purpose systems to conduct a thorough performance comparison. Results show a notable difference in times needed for processing the GSCV hyperparameter tuning method on the KNN algorithm between the proposed architecture and the general-purpose configurations.

Section II provides a careful study of earlier related papers that worked on different hardware accelerator architectures to accelerate ML algorithms. Section III describes the implementation of our unique hardware accelerator architecture in detail. Section IV presents a comprehensive performance comparison between the proposed architecture and general-purpose systems, evaluating elapsed execution times across various GSCV configurations. Section V concludes all comparisons.

2 Related works

To the best of our knowledge, no prior research has investigated the acceleration of the GSCV-KNN method using FPGA architecture. Although GSCV is widely adopted for hyperparameter tuning, its hardware-level optimization—particularly with FPGAs—remains unexplored. This gap may stem from several factors, including the dominance of software-centric optimization approaches, limited awareness of FPGA potential in ML tasks, and the integration challenges between ML frameworks and reconfigurable hardware platforms. Thus, in this section, we review studies that have accelerated other ML algorithms, including KNN, RF, SVM, Naive Bayes (NB), and Neural Networks. We also highlight the importance of GSCV to support the idea that accelerating it is worth further investigation. The findings from these studies back up our theory and align with the existing research.

2.1 Hardware Accelerators

As the volume of data generation increases every day, computers need more computing power and resources to be able to process them in an efficient and useful manner. According to many recent studies, ordinary general-purpose systems are not functional enough to handle this enormous amount of data [11]. Thus, using hardware accelerators has become very popular among ML developers, especially when their model must process a high-dimensional dataset. Several studies have explored FPGA-based architectures for accelerating various ML algorithms. In paper [1], a ubiquitous accelerator was implemented on an FPGA platform, demonstrating significant performance and energy efficiency improvements over GPU and CPU-based systems. In paper [8], an FPGA-accelerated KNN algorithm showed up to 80x speed gains and 83 percent power reduction compared to state-of-the-art CPU implementations. Paper [10] presented an FPGA solution for NB classification, achieving up to 16x speedup over an embedded ARM processor. In paper [12], FPGA-based acceleration was found to be 33646 times faster than traditional CPU implementations of Artificial Neural Network (ANN), highlighting FPGA advantages in scalability and reconfigurability. In paper [13], real-time object detection using YOLOv5 on an FPGA demonstrated doubled efficiency and reduced prediction time. Paper [14] compared FPGA performance on multiple ML algorithms, showing a 17x improvement over IoT platforms. Paper

[15] analyzed the benefits of FPGA acceleration for gradient-boosted Decision Tree (DT), achieving 2x performance gains while reducing energy consumption by 72x. In paper [16], a layer-based FPGA accelerator for Convolutional Neural Network (CNN) improved memory bandwidth utilization by 24 percent. Paper [17] showed FPGA acceleration for DT, achieving 2.48x speedup in training. In paper [18], FPGA solutions addressed network congestion, demonstrating superior performance and energy efficiency compared to CPU and GPU platforms. Paper [19] presented an FPGA-based K-means clustering accelerator, achieving up to 16.7x faster execution times. In paper [20], FPGA-accelerated SVM reduced execution time by 0.6 seconds. According to the studied papers, FPGA-based accelerators have demonstrated significant advantages in performance, scalability, and energy efficiency across a wide range of ML applications. These studies highlight the growing potential of FPGA solutions in addressing computational challenges, offering promising alternatives to traditional GPU and CPU-based approaches.

2.2 Grid Search Cross Validation method

In recent years, the importance and impact of AI and ML in our lives have been extraordinary. Nowadays, humans can carry out many amazing tasks, such as classifying massive datasets and predicting things, thanks to advances in AI and ML. Despite all these significant capabilities, AI and ML algorithms still need to become more powerful and accurate due to the exponential growth of data produced daily. Thus, methods such as data pre-processing, feature selection, and HP tuning are worth further investigation. In paper [2], a grid-search-based HP tuning approach for the RF algorithm was offered to classify microarray cancer data, yielding improvements in performance and precision. In paper [3], various HP tuning techniques—including the exhaustive GSCV method—were applied to ML models for Arabic sentiment classification, leading to enhanced accuracies despite the GSCV's high computational cost. In paper [6], GSCV was used to optimize the HPs of eight models for predicting HIV test outcomes, demonstrating that proper HP tuning can significantly impact accuracy, though its exponential complexity confines its practicality to smaller search spaces. In paper [21], employing GSCV on an SVM for DNA sequence classification boosted accuracy from 77 percent to 90 percent. In paper [5], a stacking classification framework for predicting geological characteristics achieved an accuracy of 0.996 by selecting optimal HPs through GSCV, outperforming three other tested classifiers. In paper [22], integrating ANN with GSCV for predicting indoor physical variables resulted in performance improvements over the raw ANN. In paper [7], a comparison among Grid Search, Random Search, and Genetic Algorithms revealed that all methods require considerable time for optimal model selection, with Grid Search being particularly time-consuming due to its brute-force exploration of large hyperparameter spaces. While several alternative hyperparameter optimization techniques have demonstrated improved performance over GridSearchCV in various scenarios, this paper specifically focuses on accelerating GridSearchCV due to its simplicity and widespread use among ML practitioners. By enhancing its computational efficiency, we aim to preserve its familiar workflow while making it more practical for large-scale applications. Future research may extend these acceleration strategies to other optimization methods. In conclusion, these studies demonstrate that advanced HP tuning techniques, particularly GSCV, can significantly enhance ML model accuracy and performance across diverse applications. However, the trade-off between comprehensive search capabilities and reduced speed may become a substantial limitation when working with large HP spaces.

2.3 Motivation

Papers [4], [23–25] have highlighted the significance of the KNN algorithm as a key method for classification tasks. While the algorithm is highly effective, it has a notable drawback: its sensitivity to HPs. One critical HP, which is called 'K' (or the number of nearest neighbors), directly influences the accuracy of the KNN algorithm. To address this issue, GSCV has been employed to identify the optimal value for the 'K', finally enhancing the algorithm's performance. However, as reviewed in this section, GSCV suffers from time-consuming process, necessitating efforts to accelerate its operation. In this paper, we propose an FPGA-based hardware accelerator architecture to accelerate the GSCV method while executing the HP tuning process for the KNN algorithm. This architecture aims to accelerate the process of identifying the best possible 'K' for the KNN algorithm while maintaining accuracy and efficiency.

3 Methodology

In this section, we begin by analyzing the fundamental concepts of the GSCV method and the KNN algorithm. Next, we provide a detailed explanation of our hardware accelerator architecture, including its design principles and operational framework. This discussion covers each hardware component thoroughly, offering an in-depth understanding of their functionality.

3.1 K-Nearest Neighbor Algorithm

The KNN algorithm is one of the simplest yet effective ML techniques for classification tasks. It classifies new data points with simple computational steps, making it highly intuitive and efficient [26, 27]. As illustrated in Figure 1, when a new instance (green) enters, the algorithm calculates the distance between this point and all previously classified instances (red, blue and yellow). Various distance metrics, such as Euclidean, Minkowski, and Manhattan, can be applied depending on the project requirements [28].

Once the distances are computed, Based on Figure 2, they are sorted in ascending order using a chosen sorting algorithm and stored in a list. Subsequently, the first K-instances from the sorted list are selected as the nearest neighbors. The new instance is then assigned a class based on the majority class of these K-neighbors, ensuring an intuitive and adaptable classification process.

3.2 GSCV Workflow

The GSCV method is well-known for its effectiveness and simplicity. This technique automatically selects the best HPs and eventually increases the accuracy [25]. The

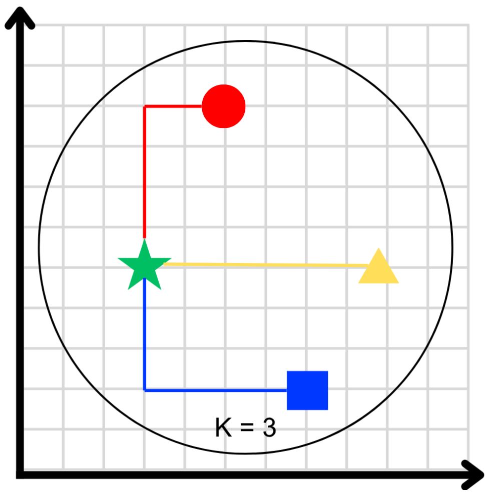

Fig. 1 KNN algorithm distance calculation process

GSCV HP selection and tuning workflow includes two separate parts, Grid-Search (GS) and Cross-validation (CV) [21]. Assume that we have two HPs with these values: X: 1,2,3,4, Y: 1,2,3. Initially, as Figure 3 demonstrates, the GS method creates a full-mesh grid space [7] with the size of (X*Y), where each node of it represents a unique combination of (X,Y). Afterward, the GS performs a complete search among all nodes, applying them one by one to the under-test ML algorithm to create unique ML models. Then, GS evaluates all created model on the dataset while considering their accuracy scores [29]. Ultimately, GS reports the best possible combination of HPs, enabling the under-test ML algorithm to achieve the highest possible accuracy on the particular dataset.

There is a problem with the raw GS method, which is over-fitting on the under-test dataset. To tackle this issue, GS is usually combined with CV [5]. The CV is commonly used to systematically vary the validation set within the training data, which helps mitigate overfitting by providing a more robust estimate of model performance across different data splits. According to Figure 4, the CV method begins by splitting the dataset into K-folds. First, it selects one fold as the test data while using the remaining k-1 folds as training data. This initial setup is referred to as ”Round 1”. Next, the classifier model is evaluated using the GS technique on ”Round 1”, and the accuracy is recorded. The CV process then rearranges the folds and repeats these steps for multiple rounds until ”Round i”. After completing all rounds, the method calculates the average of the recorded accuracies and reports this value as the overall accuracy of the created model for a given set of HP combinations. Finally, the classifier's parameters are adjusted through GS, and the accuracy is reevaluated [22]. Consequently, all mentioned steps are combined together and create the GSCV method, which is used for tuning ML algorithms' HPs.

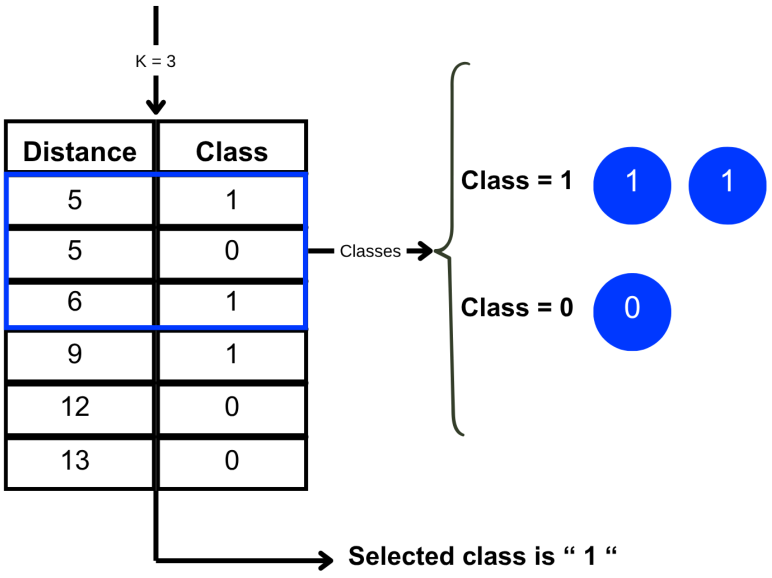

Fig. 2 KNN algorithm class selection process

3.3 Combination of GSCV and KNN

Nearly all ML algorithms include 2 different types of parameters [3].

1. Model parameter: ML algorithms automatically adjust these internal parameters based on the working dataset.
2. Hyper parameter: These parameters should be configured before the training phase in order to achieve the desired result.

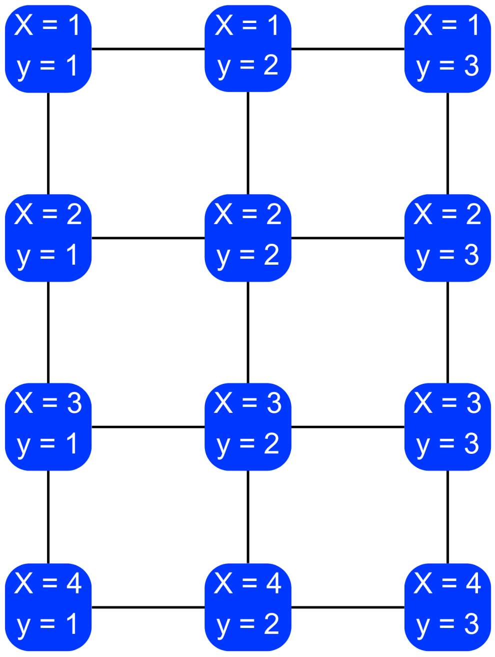

Fig. 3 The GridSearch workflow

According to previous research, tuning HPs will increase the accuracy and performance of ML algorithms [29]. Therefore, GSCV can be applied to all ML models to automatically tune and select the best HPs for them and eventually enhance their functionality. When it comes to the KNN algorithm, HP 'K' or K-nearest neighbor is highlighted more than other HPs. It is because 'K' has a direct considerable impact on the KNN output and accuracy [4]. As Figure 5 shows, different values of 'K' achieve distinct calcification outputs for new data. while Value (K=3) classifies new data as red, Value (K=8) classifies it as blue, and it's completely different. Thus, 'K' is the most effective HP for the KNN algorithm. Nevertheless, this algorithm includes other HPs as well as 'K' [25], such as:

- Weights: closer points can have greater influence than the points that are farther away
- Algorithm: There are different algorithms used to compute the nearest neighbors
- Metric: Which metric shall be used to calculate the distance in the algorithm

Consequently, selecting the best combination among all these HPs is burdensome and needs an immediate solution. In this paper, we initially employ the GSCV method to address the challenge of hyperparameter selection in the KNN model. As Figure 6 illustrates a flowchart, GSCV facilitates the systematic exploration of hyperparameter combinations, enabling the identification of the most effective configuration to improve model accuracy. Although GSCV can be effective, its practical application is limited

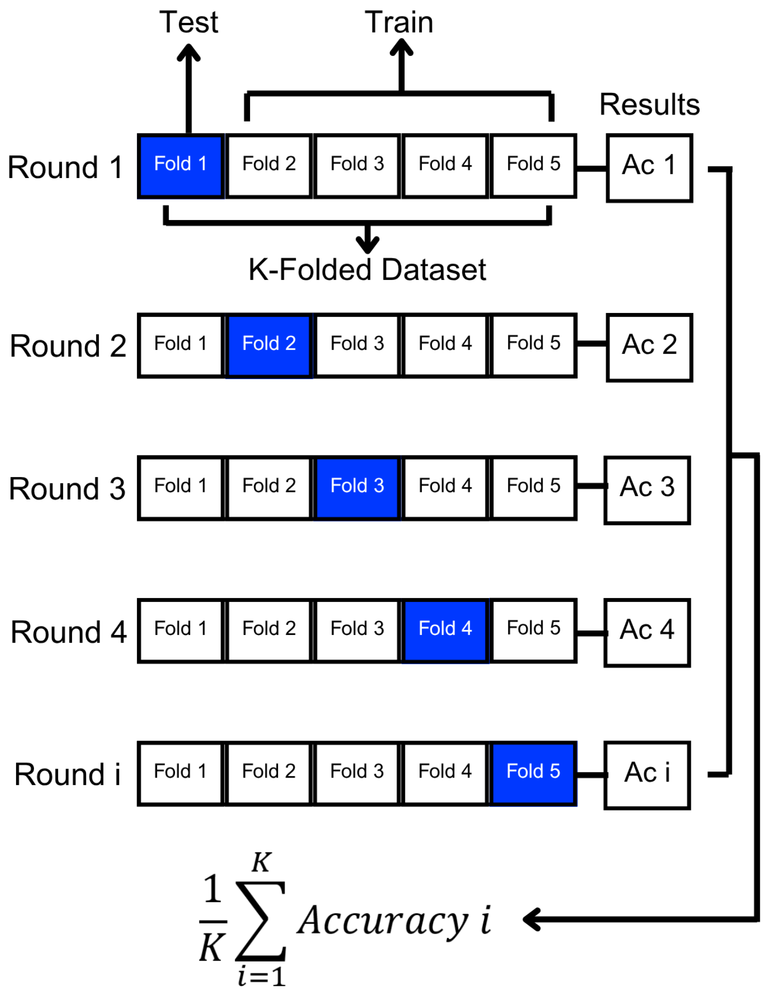

Fig. 4 K-fold Cross-validation workflow

by high computational cost, which is documented in the prior works [4, 6, 7]. The method's exhaustive GS scales multiplicatively with the number of hyperparameter combinations, the number of CV folds, and the dataset size. For example, consider three hyperparameters X, Y, and Z with 4, 2, and 3 candidate values respectively, producing

$$4 \cdot 2 \cdot 3 = 24$$

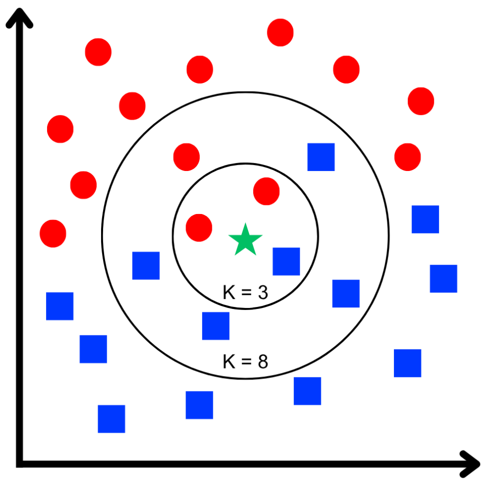

Fig. 5 impact of hyperparameter 'K' on the KNN

grid points. Using 10-fold CV on a dataset with $N = 500{,}000$ samples yields on the order of

$$24 \cdot 10 \cdot N \approx 120 \cdot 10^6$$

sample–evaluations. If the average time to process a single sample during one evaluation is τ seconds, the total wall-clock time is approximately

$$120 \cdot 10^6 \cdot \tau.$$

With a plausible per-sample cost of $\tau = 1$ ms, this corresponds to ≈ 33.3 hours of computation for a single tuning run. Such costs grow rapidly with larger grids, more folds, or larger datasets, motivating the need for acceleration. To address this bottleneck, we propose an FPGA-based hardware accelerator designed to substantially reduce evaluation latency and to support flexible choices of K-fold values that can improve validation granularity and model-selection accuracy.

3.4 Hardware Accelerator

This paper presents a unique FPGA-based hardware accelerator, which is designed to accelerate the GSCV method while performing HP tuning for the KNN algorithm. The proposed architecture is simulated and implemented directly in VHDL, without relying on any code conversion frameworks. To achieve this, we first deconstructed the KNN algorithm workflow into smaller components and identified that KNN consists of two computationally intensive operations, 'distance calculation' and 'sorting distances' [23]. Operation 'distance calculation' was about calculating distances between the new data and all other classified data. Additionally, Operation 'sorting distances' was about sorting the calculated distances in ascending order for further operations such as 'class detection'. Secondly, we performed the same thing on the GSCV algorithm and deconstructed it into smaller operations. We found out that GSCV had two

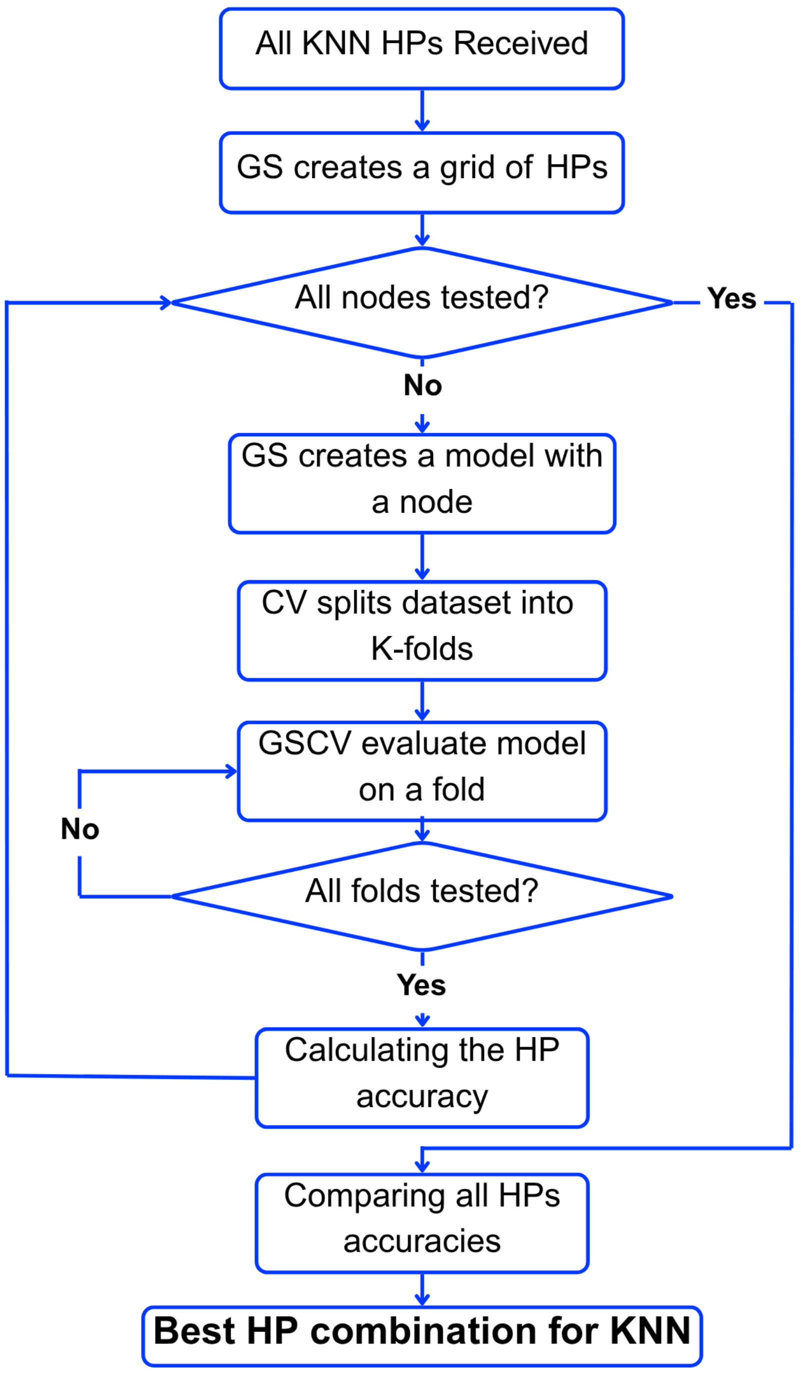
All KNN HPs Received
GS creates a grid of HPs
All nodes tested?
Yes
No
GS creates a model with a node
CV splits dataset into K-folds
GSCV evaluate model on a fold
No
All folds tested?
Yes
Calculating the HP accuracy
Comparing all HPs accuracies
Best HP combination for KNN

Fig. 6 Flowchart of using GSCV on KNN

computationally intensive operations as well, 'K-fold data division' and 'HP accuracy calculation'. According to their names, they are about dividing the dataset into K-folds, controlling the whole data stream throughout the architecture, and calculating HP's average accuracy based on the folds' results. Finally, after a detailed analysis of GSCV-KNN operations, we designed a core architecture to execute GSCV on KNN, as shown in Figure 7.

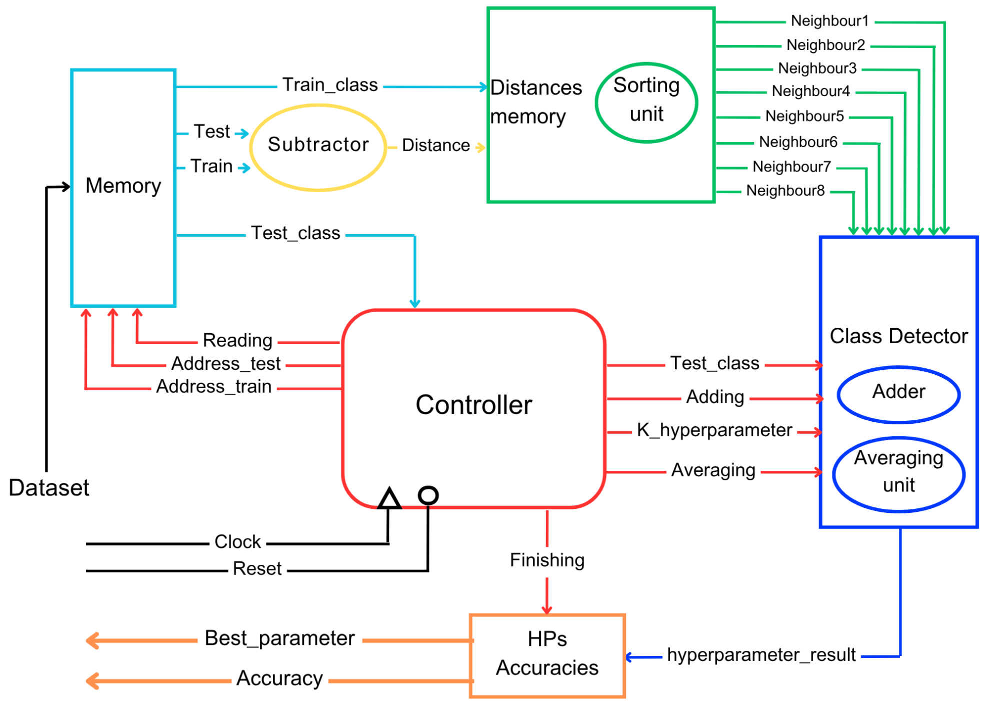

Fig. 7 The proposed FPGA-based accelerator architecture

As Figure 7 demonstrates, our fundamental architecture contains 6 modules. 1. Main Memory: We designed a custom on-chip RAM array, which holds the entire dataset without relying on external DRAM. The memory is organized as a $253{,}680 \times 22$ dual-port array—one row per sample, 22 feature words per row—for a total of 5,580,960 storage words. During the system's reset phase, all dataset values are loaded into this RAM via a lightweight configuration interface; after this one-time, bulk transfer, the data remain on-chip for the entire GSCV–KNN procedure. By eliminating repeated accesses to off-chip memory, the design avoids long external-DRAM latencies and sustains a continuous, high-throughput processing pipeline, markedly improving both speed and energy efficiency. At runtime, the Control Unit drives both read ports on every 10ns clock edge, issuing two independent read addresses and enable signals simultaneously. Each port returns its full 22-word sample in a single cycle, enabling the accelerator to fetch and process two samples per cycle. Before computation begins, the Control Unit also partitions the in-RAM dataset into K folds by assigning contiguous row ranges to each fold; this on-chip folding requires no extra software steps and ensures that every cross-validation iteration reads from the correct subset. The result is deterministic, two-sample access per cycle, a perfectly balanced pipeline flow, and minimal control overhead—key factors that allow our FPGA design to meet the extreme computational and memory demands of the 'CV-full' workload. 2. Subtractor: This module accepts two inputs—a training instance and a test instance—and computes the distance between them using the Manhattan distance metric. Selecting the appropriate distance metric posed a challenge, as each has its own strengths and limitations depending on the task at hand [26, 28, 30, 31]. Nevertheless, Manhattan distance has demonstrated strong performance in KNN applications [32], and is considered a suitable choice for use in integrated circuit contexts [33]. The subtractor component is built with a simple and flexible architecture, enabling straightforward substitution of alternative metric functions such as Euclidean or Minkowski. Only the core logic responsible for distance computation requires modification; the rest of the component remains unaffected. Additionally, the Distance Memory and Class Detector modules work independently from the metric function logic, ensuring they operate correctly even when switching from Manhattan to Euclidean or Minkowski metrics. As shown in Equation 1, the Manhattan metric calculates the sum of the absolute differences between corresponding Cartesian coordinates.

$$MD(x, y) = \sum_{i=1}^{n} |x_i - y_i| \tag{1}$$

3. Distance Memory: The Distance Memory unit is designed to maintain the eight smallest distances in real time using an eight-slot register composed of D-type flip-flops. As each new distance is computed, it is immediately compared with existing values and inserted into the appropriate position through a hardware-based comparator network. This inline insertion mechanism ensures that the register remains continuously sorted without requiring separate sorting stages or iterative algorithms such as Bubble Sort or Merge Sort [34, 35]. The insertion and comparison operations are fully pipelined and occur concurrently with distance computation, introducing no additional processing delay. This architecture enables the eight nearest neighbors to be available at each clock cycle, supporting real-time performance with high throughput. We selected (K=8) as a practical implementation parameter, reflecting its common usage in typical KNN applications. However, the architecture is readily scalable to ten or more flip-flops without affecting correctness or timing, ensuring flexibility for broader use cases. 4. Class Detector: This module performs two crucial tasks in class detection. First, it assigns a class to the test instance based on the nearest neighbors identified in the Distance Memory component. The HP 'K' determines how many nearest neighbors should contribute to the detection process. For example, if (K=3), only the three closest neighbors of the test instance are considered. Next, the Class Detector compares the assigned class with the actual class of the test instance, which was previously stored in the Main Memory. The outcome of this comparison—either 1 (match) or 0 (mismatch)—is recorded. Once all folds have been evaluated, the total

number of matches is divided by the total number of tested instances to determine the accuracy of the HP combination. 5. HPs Accuracies: All calculated HP combination accuracies are saved here, and when the signal 'Finish' comes, this component reports the best possible HP combination. 6. Controller: This component collects all HPs and constructs a grid, based on various HP combinations. Additionally, the Controller executes K-fold CV, oversees data flow, and generates and issues signals for other components in order to work synchronously and correctly by every positive edge of the FPGA clock.

3.5 Implementation

Our implementation consists of two distinct approaches: software-based and hardware-based. In the software-based approach, we implement GSCV-KNN in a software environment and execute it on several general-purpose computers. Meanwhile, in the hardware-based approach, we integrate our proposed GSCV-KNN architecture within a hardware simulation environment, systematically recording all constraints.

- Software-based implementation: To implement the GSCV-KNN method in software, we used Python with Scikit-learn, Pandas, and NumPy. In particular, Scikit-learn[36] provided the baseline implementations for both the KNN classifier and the GSCV routine; the complete program code is shown below. We executed our developed GSCV-KNN core on seven general-purpose systems, only a subset of which featured dedicated GPUs. However, every experiment—whether run on a CPU-only machine or a GPU-equipped one—invoked the identical pure-CPU Scikit-learn code. We did not integrate custom CUDA kernels, PyCUDA bindings, or any other low-level GPU programming, and thus no GPU resources were leveraged during those runs. This design choice enforces a consistent, fair baseline comparison against our FPGA accelerator without confounding results with GPU-accelerated optimizations. We acknowledge that GPU-optimized libraries such as RAPIDS cuML or FAISS—both of which offer zero-code-change integration with Scikit-learn—can deliver substantial performance gains via parallelism and efficient memory handling; benchmarking against these frameworks is reserved for future work to more comprehensively evaluate the scalability and efficiency of our hardware solution.

```
# Import necessary libraries
  from sklearn.model_selection import GridSearchCV
  from sklearn.neighbors import KNeighborsClassifier
# Define the hyperparameter grid
  param_grid =
  {
     'n_neighbors': list(range(1, 9)),  # Number of neighbors from 1 to 8
     'algorithm': ['brute'],            # Brute-force search
     'metric': ['manhattan']            # Manhattan distance metric
  }
# Initialize KNN classifier
  knn = KNeighborsClassifier()
```

```
# Initialize GridSearchCV with KNN and parameter grid
  grid_search = GridSearchCV(estimator=knn, param_grid=param_grid)
# Fit the GridSearchCV model
  grid_search.fit(X_train, y_train)
# Get the best parameters
  best_params = grid_search.best_params_
  print("Best Hyperparameters:", best_params)
```

- Hardware-based implementation: To implement the GSCV-KNN accelerator, the design was described in VHDL, functionally verified using behavioral and post-synthesis simulations in ModelSim, and synthesized with Intel Quartus targeting a 'Cyclone V' FPGA device. Post-synthesis simulations were performed using a 10ns clock period (100 MHz) to evaluate performance, and the resulting resource utilization report is presented in Table 1. These results indicate that the design occupies only a small fraction of the target device's resources. Due to limited access to physical measurement tools and on-board testing equipment, all reported performance, timing, and resource metrics are derived from simulation. Nevertheless, the use of industry-standard tools such as Quartus provides reliable estimates that closely approximate real-world behavior. The primary focus of this work is to accelerate the execution time of GridSearchCV-KNN using FPGA-based architecture; power consumption and energy efficiency, while important, are beyond the scope of this study. Future work may include comprehensive on-board validation and broader cross-platform comparisons involving additional performance metrics.

Table 1 Post-synthesis resource utilization report

Item	Report
Flow Status	Successful - Apr.15.2025
Top-level Entity Name	GSCV
Family	Cyclone IV GX
Total logic elements	1,246 / 21,280 (6 %)
Total combinational functions	1,237 / 21,280 (6 %)
Dedicated logic registers	150 / 21,280 (¡ 1 %)
Total registers	150
Total pins	65 / 167 (39 %)
Total virtual pins	0
Total memory bits	0 / 774,144 (0 %)
Total PLLs	0 / 4 (0 %)
Device	EP4CGX22CF19C6

4 Results and Discussion

In this section, we compare the recorded elapsed times of executing GSCV-KNN. These elapsed times are gathered from 7 distinct general-purpose systems and our proposed architecture. The configuration of the different tasted systems is shown in Table 2. The

Table 2 Configurations of the systems under test

Systems	CPU	GPU
PC	Intel Core i3-12100	-
Laptop 1	Intel Core i7-4500U	-
Laptop 2	AMD Ryzen-9 4900HS	NVIDIA GeForce RTX-2060
Laptop 3	Intel Core i7-6500U	NVIDIA GeForce GTX-1060
Google Colab Server 1	Intel Xeon-79	-
Google Colab Server 2	Intel Xeon Gold-63	NVIDIA Tesla-T4 Tensor Core
Ferdowsi University Server	AMD EPYC-7763	NVIDIA Tesla-V100

utilized dataset for the testing process is the 'Diabetes Health Indicators dataset' [37], from the UCI ML repository, which contains comprehensive healthcare statistics and lifestyle survey information. It provides data on individuals' health profiles, including their diabetes diagnosis categorized as diabetic, pre-diabetic, or healthy. This dataset was selected due to its high dimensionality and large volume of instances (253680 rows), making it computationally intensive yet proper for testing different computing systems' performance.

The GSCV method has several configurable parameters, with 'CV' determining the number of folds used during the tuning process. This parameter directly impacts both the complexity and accuracy of GSCV. To expand our experiments, we tested different values of 'CV', selecting 5 and 10 as they are commonly used. To further amplify the time complexity of software-based implementations, we incorporated CV-full (equivalent to each instance being one fold), ensuring a more demanding computational challenge. This CV-full comes from Leave One Out Cross Validation (LOOCV). The LOOCV is a cross-validation algorithm which is a more extreme version of k-fold cross-validation. The number of folds in a LOOCV equals the amount of data, just like our CV-full value. In LOOCV, the dataset is split into multiple training and testing sets by systematically leaving out one observation at a time. For each iteration, the model is trained on all data points except one, which is then used as the test case. This process is repeated for every data point, resulting in K training-testing cycles for a dataset with K observations. The final performance metric is computed by averaging the errors across all iterations. The LOOCV offers a unbiased and more accurate estimate of model performance than other CV values since each data point is used for testing exactly once, but it can be computationally intensive for large datasets [38, 39]. Therefore, the CV-full was chosen not only because our proposed FPGA-based architecture is designed to operate specifically with this particular value, but also by implementing CV-full, we aimed to evaluate our architecture under the most challenging conditions. It is worth mentioning that some of the systems could not execute CV-full due to a shortage of computational capability, so we changed their CV-full to CV-half (equivalent to two instances being one fold). Despite these changes, our FPGA-based solution has consistently demonstrated the fastest performance across all tested configurations. Figure 8 illustrates the performance comparison between our architecture and the Intel Core i7-4500U across various CV values. In this and all similar figures, the Y-axis represents the execution time of the GSCV-KNN algorithm under different CV configuration, while the X-axis denotes the number of dataset rows used in the experiments. Figure 8 clearly demonstrates the substantial performance advantage of our architecture over the Intel Core i7-4500U. Figure 9 presents a com-

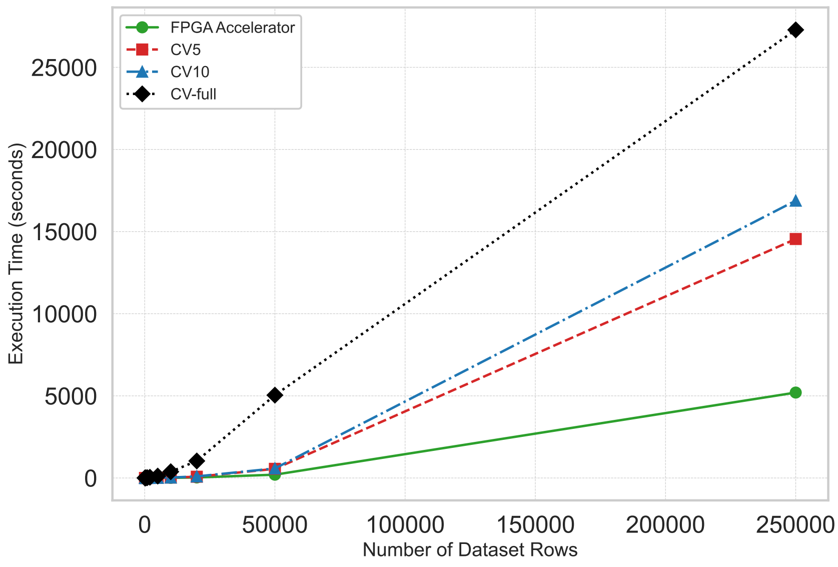

Fig. 8 Performance Comparison Between FPGA and Intel Core I7-4500U

parative analysis of our FPGA-based architecture and a Google Colab server equipped with an Intel Xeon E7-79 CPU (without GPU acceleration). In this and all analogous figures, the Y-axis represents the execution time of the GSCV-KNN algorithm, while the X-axis corresponds to the number of dataset samples. When the dataset grows beyond 50000 samples, the CV-full run on the Colab server shows a sharp spike in runtime, while our FPGA design continues to scale gracefully. This inflection point appears in every chart from Figures 8 to 14: below 50000, the GSCV-KNN workload is light enough that FPGA and other systems' runtimes remain comparable; above it, the quadratic cost of distance calculations and off-chip data movement amplifies each architecture's strengths and weaknesses. By leveraging a deeply pipelined dataflow, dedicated special-purpose compute units, and unique on-chip buffering, the FPGA handles the increased computational load with minimal overhead—achieving a distinct performance edge at and beyond the 50000-sample mark. According to Figure 10, the Intel Core i3-12100 demonstrated strong performance with a small dataset. However, when tested on the full dataset, it was unable to surpass our FPGA accelerator. Figure 11 compares the FPGA accelerator and a system consisting of a Core i7-6500U CPU paired with an NVIDIA GeForce GTX-1060 GPU. Figure 12 compares the performance of our FPGA-based accelerator with an AMD Ryzen-9 4900HS CPU. Although the Ryzen-9 4900HS delivers strong results, our accelerator consistently outperforms

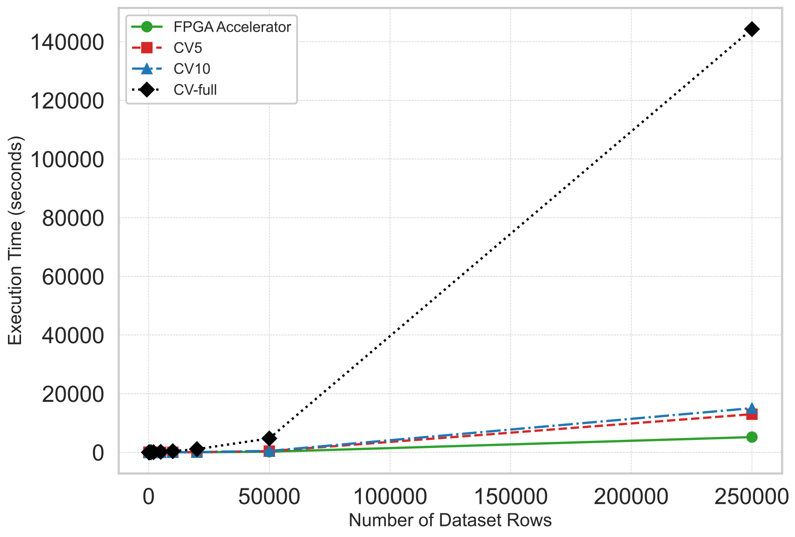

Fig. 9 Performance Comparison Between FPGA and Intel Xeon-79 CPU

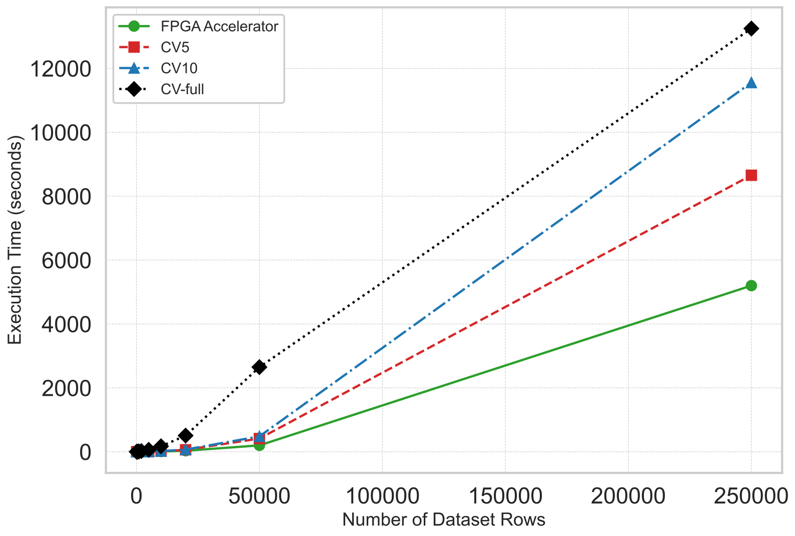

Fig. 10 Performance Comparison Between FPGA and Intel Core i3-12100 CPU

the CPU, especially when the GSCV-KNN configuration is set to its maximum values. Figure 13 demonstrates the performance advantage of our FPGA-based architecture over the Google Colab server environment, which is one of the most widely used platforms for developing and executing ML and AI workloads. The results confirm that our system consistently outperforms this common baseline. Figure 14 compares the performance of our FPGA accelerator with a high-performance server featuring an AMD EPYC 7763 CPU paired with an NVIDIA Tesla V100 GPU. As the number

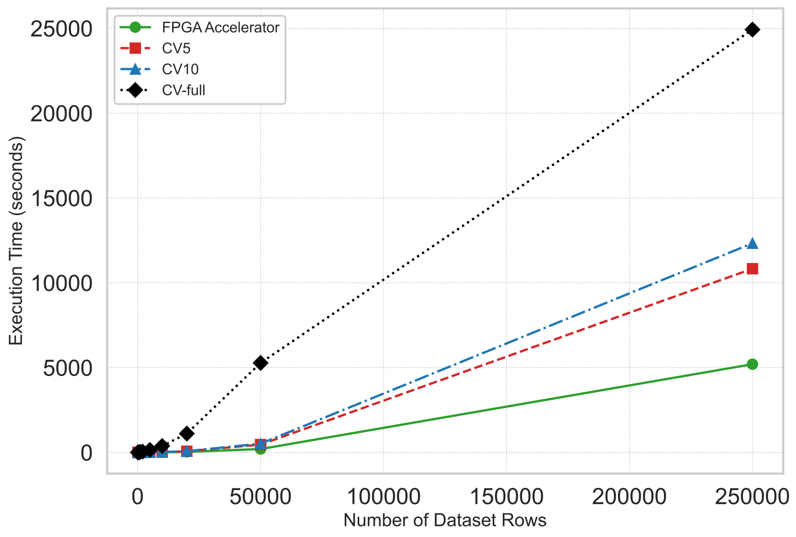

Fig. 11 Performance Comparison Between FPGA and Intel Core i7-6500U CPU

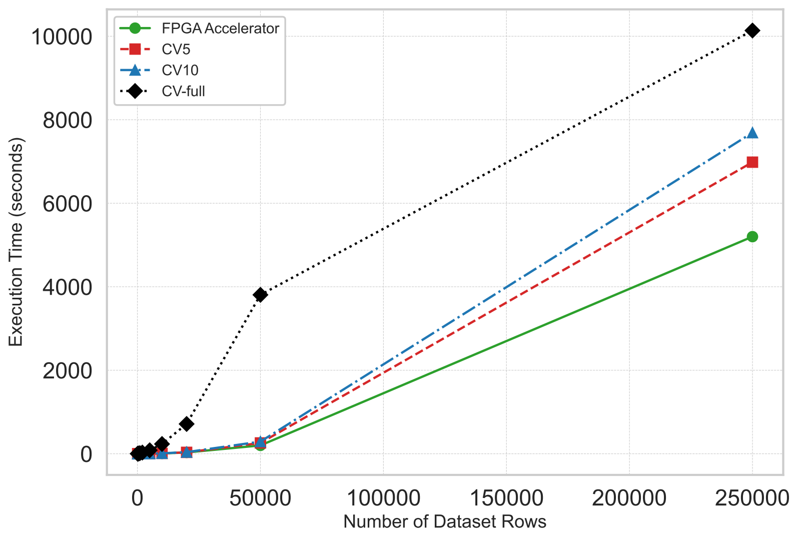

Fig. 12 Performance Comparison Between FPGA and AMD Ryzen-9 4900HS CPU

of dataset samples grows or the CV configuration reaches its maximum settings, the performance gap further widens, consistent with the trends shown in previous figures.

In Figures 15, 16, and 17, we compare all tested systems' performances alongside our proposed FPGA architecture, focusing on the impact of different CV values. Value 5 represents the least complexity, while Full indicates the highest. According to the data presented in the charts, our architecture consistently achieves superior performance in terms of elapsed execution time for GSCV-KNN, outperforming all

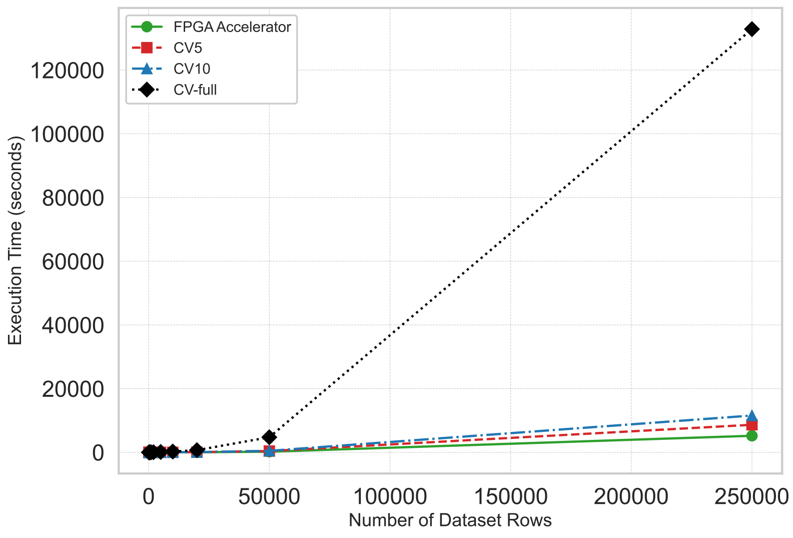

Fig. 13 Performance Comparison Between FPGA and NVIDIA Intel Xeon Gold-63 CPU

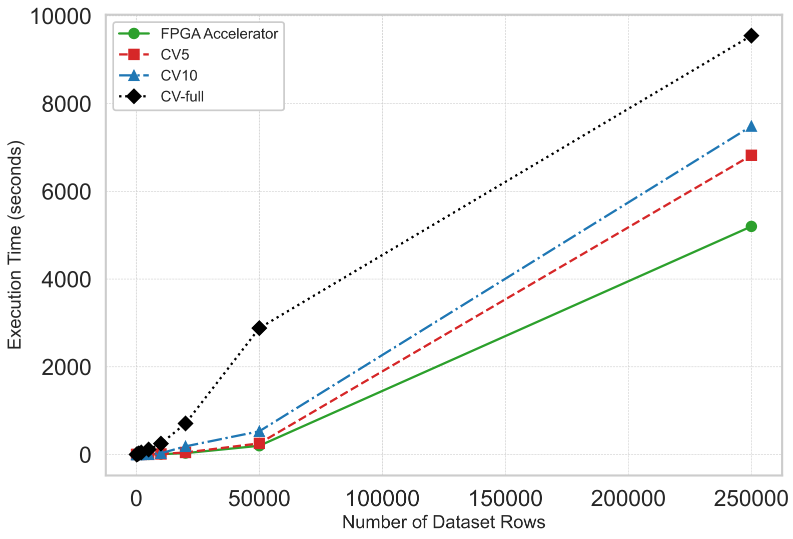

Fig. 14 Performance Comparison Between FPGA and AMD EPYC-7763 CPU

other systems across different CV values, from the simplest to the most complex. Our FPGA accelerator was implemented exclusively for the CV-full cross-validation mode—the most demanding configuration, in which each dataset row serves as its own fold, yielding 253680 iterations and imposing extreme computational and memory demands. We measured runtime via deterministic post-synthesis timing simulation under a fixed 10ns clock period. Because the design does not natively support other

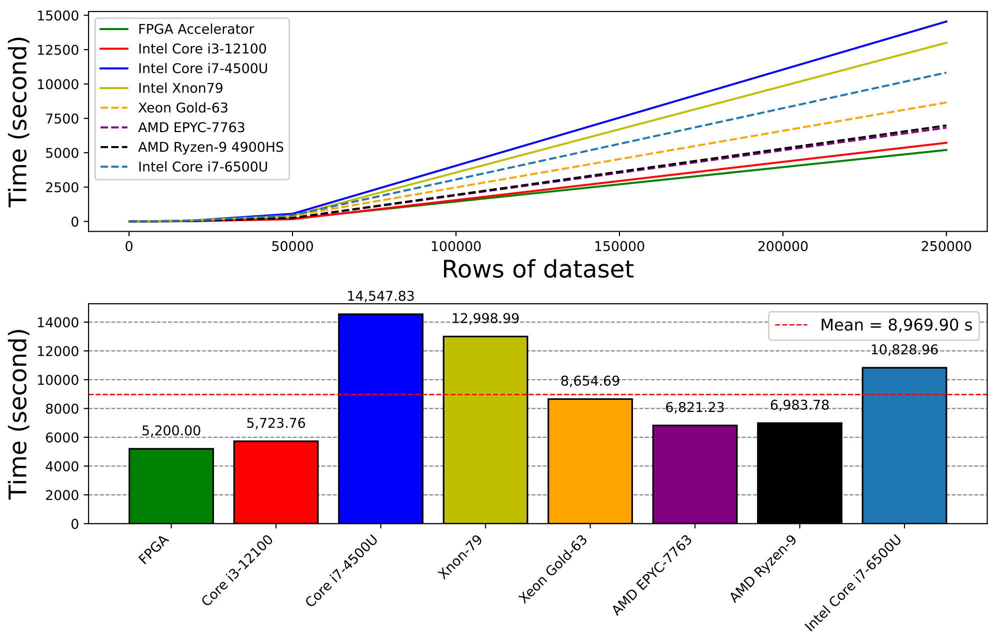

Fig. 15 Performance of All Systems with CV Set to 5

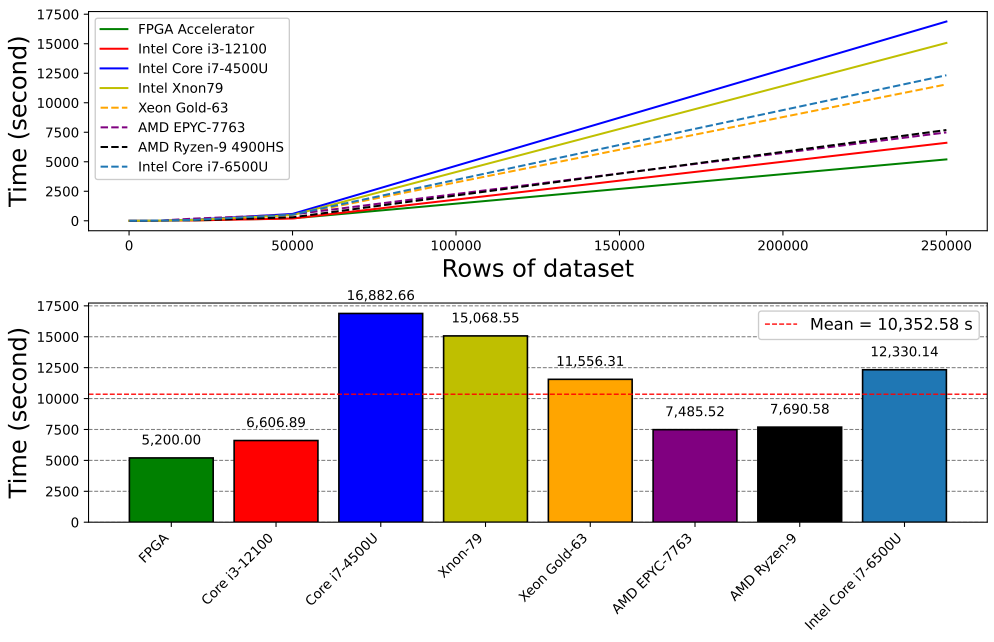

Fig. 16 Performance of All Systems with CV Set to 10

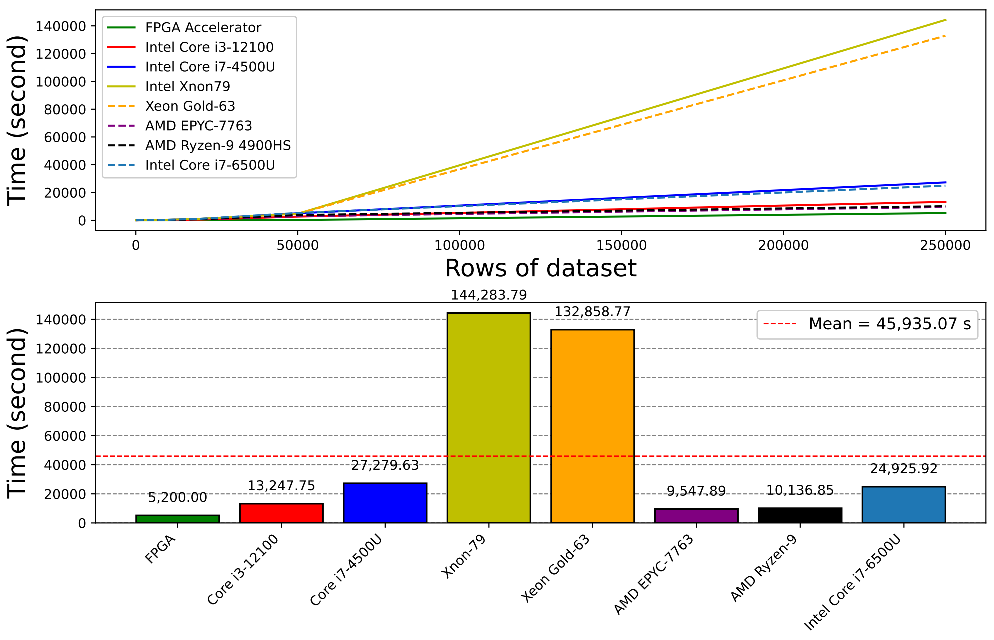

Fig. 17 Performance of All Systems with CV Set to full

k-fold modes, we conservatively applied the CV-full timing to represent the less intensive CV-5 and CV-10 scenarios in Table 3 and Table 4. This strategy ensures a fair, albeit conservative, comparison with prior work that typically reports CV-5 or CV-10 results.

The uniformity of the FPGA execution times in Table 3 to 5 arises from two factors: a single, repeatable post-synthesis timing simulation of the CV-full workload and a fully defined processing pipeline running at a fixed 10ns clock, which yields a deterministic cycle count and thus a constant wall-clock time. Despite the CV-full workload's intensity, our FPGA design outperforms the average of all evaluated general-purpose systems by factors of 1.8 for CV-5, 2.12 for CV-10, and 9.92 for CV-full, as its shown in Table 6. Moreover, several general-purpose systems failed to complete the CV-full run due to insufficient main memory, underscoring the efficiency and robustness of the proposed accelerator.

Table 3 presents the elapsed times for GSCV-KNN with CV-5 configuration, measured on our FPGA accelerator and seven general-purpose computing systems. To evaluate performance gains, the recorded times from each system were divided by the FPGA's elapsed time. As shown in Table 6, the FPGA accelerator achieves an average speedup of approximately 2× for CV-5 compared to the other platforms.

Table 4 illustrates the performance improvement achieved by our FPGA accelerator during the execution of the GSCV method for HP tuning of the KNN algorithm. For each system, the enhancement was calculated by dividing its elapsed time by that

Table 3 FPGA performance enhancement for CV-5

Processors	Elapsed time for CV-5	FPGA performance enhancement
Proposed FPGA Architecture	5200 s	-
Intel core i3-12100	5723.76 s	1.1x
Intel core i7-4500	14547.83 s	2.7x
Intel Xeon-79	12988.99 s	2.49x
AMD EPYC-7763	6821.23 s	1.31x
AMD Ryzen-9 4900HS	6983.78 s	1.34x
Intel Xeon Gold-63	8654.69 s	1.66x
Intel Core i7-6500U	10828.96 s	2.08x

of the FPGA. As summarized in Table 6, the FPGA accelerator delivers an average speedup of 2.12× for ten-fold cross-validation CV-10 relative to the other platforms.

Table 4 FPGA performance enhancement for CV-10

Processors	Elapsed time for CV-10	FPGA performance enhancement
Proposed FPGA Architecture	5200 s	-
Intel core i3-12100	6606.89 s	1.27x
Intel core i7-4500	16882.66 s	3.24x
Intel Xeon-79	15068.55 s	2.89x
AMD EPYC-7763	7485.52 s	1.43x
AMD Ryzen-9 4900HS	7690.58 s	1.47x
Intel Xeon Gold-63	11556.31 s	2.22x
Intel Core i7-6500U	12330.14 s	2.37x

Table 5 reports the performance enhancement achieved by our FPGA accelerator when executing the GSCV method for hyperparameter tuning of the KNN algorithm under LOOCV. For each system, the speedup was calculated by dividing its elapsed time by the corresponding FPGA execution time. As indicated in Table 6, the FPGA accelerator demonstrates a substantial average speedup of approximately 10× compared to the general-purpose platforms.

Table 5 FPGA performance enhancement for CV-full

Processors	Elapsed time for CV-full	FPGA performance enhancement
Proposed FPGA Architecture	5200 s	-
Intel core i3-12100	13247.75 s	2.54x
Intel core i7-4500	27279.63 s	5.24x
Intel Xeon-79	144283.79 s	27.7x
AMD EPYC-7763	9547.89 s	1.83x
AMD Ryzen-9 4900HS	10136.85 s	1.94x
Intel Xeon Gold-63	132858.77 s	25.5x
Intel Core i7-6500U	24925.92 s	4.7x

Table 6 Average performance enhancement

performance enhancement (CV-5)	performance enhancement (CV-10)	performance enhancement (CV-full)
1.8x	2.12x	9.92x

5 Conclusion

This paper introduced a high-performance FPGA accelerator architecture designed to enhance the execution speed of the Grid Search Cross-Validation (GSCV) method during hyperparameter tuning of the K-Nearest Neighbors (KNN) algorithm. The proposed solution was benchmarked against seven diverse general-purpose computing platforms, including both CPU- and GPU-based configurations, across three cross-validation schemes: five-fold (CV-5), ten-fold (CV-10), and leave-one-out (CV-full). Experimental results demonstrated consistent and significant performance improvements. Specifically, the FPGA accelerator achieved an average speedup of approximately 2× for CV-5, 2.12× for CV-10, and an impressive 10× for CV-full, as reported in Tables 3 through 6. These findings highlight the scalability and efficiency of the proposed architecture, particularly as the computational complexity of GSCV-KNN increases with higher CV values. The substantial acceleration observed under CV-full suggests that the FPGA-based approach is especially advantageous for large-scale datasets or compute-intensive tasks, making it a promising candidate for deployment in real-world machine learning applications where speed and resource efficiency are critical.

Declarations

- Funding: 'Not applicable'
- Conflict of interest: The authors declare no conflicts of interest regarding this manuscript.
- Ethics approval and consent to participate: 'Not applicable'
- Consent for publication: 'Not applicable'
- Data availability: The dataset used in this study, the Diabetes dataset, is publicly available from the UCI Machine Learning Repository. It can be accessed at https://archive.ics.uci.edu/dataset/34/diabetes. This dataset contains records relevant to diabetes classification and prediction, which were utilized for model training and analysis. Researchers and readers can freely access and download the dataset for further studies in computational research. The dataset poses no privacy threat to patients, as it includes neither names nor any other identifiable information.
- Materials availability 'Not applicable'
- Code availability: The code used in this study is available from the corresponding author upon reasonable request. Due to project-specific constraints, direct access is not provided through public repositories. However, interested researchers

may contact the authors to obtain relevant code and implementation details for reproducibility and further analysis.

- Author contribution ‘Not applicable’

References

[1] Wang, C., Gong, L., Li, X. & Zhou, X. A ubiquitous machine learning accelerator with automatic parallelization on fpga. *IEEE Transactions on Parallel and Distributed Systems* **31**, 2346–2359 (2020). URL https://doi.org/10.1109/TPDS.2020.2990924.

[2] Shekar, B. H. & Dagnew, G. *Grid search-based hyperparameter tuning and classification of microarray cancer data*, 1–8 (Gangtok, India, 2019). URL https://doi.org/10.1109/ICACCP.2019.8882943.

[3] Elgeldawi, E., Sayed, A., Galal, A. R. & Zaki, A. M. Hyperparameter tuning for machine learning algorithms used for arabic sentiment analysis. *Informatics* **8**, 79 (2021). URL https://doi.org/10.3390/informatics8040079.

[4] Taunk, K., De, S., Verma, S. & Swetapadma, A. *A brief review of nearest neighbor algorithm for learning and classification*, 1255–1260 (Madurai, India, 2019). URL https://doi.org/10.1109/ICCS45141.2019.9065747.

[5] Yan, T., Shen, S.-L., Zhou, A. & Chen, X. Prediction of geological characteristics from shield operational parameters by integrating grid search and k-fold cross validation into a stacking classification algorithm. *Journal of Rock Mechanics and Geotechnical Engineering* **14**, 1292–1303 (2022). URL https://doi.org/10.1016/j.jrmge.2022.03.002.

[6] Belete, D. M. & Huchaiah, M. D. Grid search in hyperparameter optimization of machine learning models for prediction of hiv/aids test results. *International Journal of Computers and Applications* **44**, 875–886 (2021). URL https://doi.org/10.1080/1206212X.2021.1974663.

[7] Liashchynskyi, P. & Liashchynskyi, P. Grid search, random search, genetic algorithm: A big comparison for nas. arXiv preprint arXiv:1912.06059 (2019). URL https://doi.org/10.48550/arXiv.1912.06059.

[8] Liu, L. & Khalid, M. A. S. *Acceleration of k-nearest neighbor algorithm on fpga using intel sdk for opencl*, https://doi.org/1070–1073 (Windsor, ON, Canada, 2018). URL https://doi.org/10.1109/MWSCAS.2018.8623861.

[9] Ali, D., Rehman, A. U. & Khan, F. H. *Hardware accelerators and accelerators for machine learning*, 1–7 (Chiniot, Pakistan, 2022). URL https://doi.org/10.1109/ICIT56493.2022.9989124.

[10] Tzanos, G., Kachris, C. & Soudris, D. *Hardware acceleration on gaussian naive bayes machine learning algorithm*, 1–5 (Thessaloniki, Greece, 2019). URL https://doi.org/10.1109/MOCAST.2019.8741875.

[11] Kim, V. & Choi, K. A reconfigurable cnn-based accelerator design for fast and energy-efficient object detection system on mobile fpga. *IEEE Access* **11**, 59438–59445 (2023). URL https://doi.org/10.1109/ACCESS.2023.3285279.

[12] Itagi, A., Krishvadana, S., Bharath, K. P. & Rajesh Kumar, M. *Fpga architecture to enhance hardware acceleration for machine learning applications*, 1716–1722 (Erode, India, 2021). URL https://doi.org/10.1109/ICCMC51019.2021.9418015.

[13] Saidani, T. *et al.* Hardware acceleration for object detection using yolov5 deep learning algorithm on xilinx zynq fpga platform. *Engineering, Technology and Applied Science Research* **14**, 13066–13071 (2024). URL https://doi.org/10.48084/etasr.6761.

[14] Chen, R., Wu, T., Zheng, Y. & Ling, M. Mlof: Machine learning accelerators for the low-cost fpga platforms. *Applied Sciences* **12**, 89 (2022). URL https://doi.org/10.3390/app12010089.

[15] Alcolea, A. & Resano, J. Fpga accelerator for gradient boosting decision trees. *Electronics* **10**, 314 (2021). URL https://doi.org/10.3390/electronics10030314.

[16] Wu, C. B., Wang, C. S. & Hsiao, Y. K. *Reconfigurable hardware architecture design and implementation for ai deep learning accelerator*, 154–155 (Kobe, Japan, 2020). URL https://doi.org/10.1109/GCCE50665.2020.9291854.

[17] Zoulkarni, A., Kachris, C. & Soudris, D. *Hardware acceleration of decision tree learning algorithm*, 1–6 (Bremen, Germany, 2020). URL https://doi.org/10.1109/MOCAST49295.2020.9200255.

[18] Suresh, A., Reddy, B. N. & Madhavi, C. R. *Hardware accelerators for edge enabled machine learning*, 409–413 (Osaka, Japan, 2020). URL https://doi.org/10.1109/TENCON50793.2020.9293918.

[19] Jaballi, E., Gdaim, S. & Liouane, N. *Design and implementation of fpga-based hardware acceleration for machine learning using opencl: A case study on the k-means algorithm*, 1–6 (Paris, France, 2024). URL https://doi.org/10.1109/ICCAD60883.2024.10553964.

[20] Jakjoud, F., Hatim, A. & Abella, B. Fpga-based hardware acceleration for svm machine learning algorithm. *E3S Web of Conferences* **229**, 01024 (2021). URL https://doi.org/10.1051/e3sconf/202122901024.

[21] Mangkunegara, I. S. & Purwono, P. *Analysis of dna sequence classification using svm model with hyperparameter tuning grid search cv*, 427–432 (Malang, Indonesia, 2022). URL https://doi.org/10.1109/CyberneticsCom55287.2022.9865624.

[22] Alshammari, T. Using artificial neural networks with gridsearchcv for predicting indoor temperature in a smart home. *Engineering, Technology and Applied Science Research* **14**, 13437–13443 (2024). URL https://doi.org/10.48084/etasr.7008.

[23] Kanan, A. & Taha, A. *Cloud-based reconfigurable hardware accelerator for the knn classification algorithm*, 308–312 (Al-Khobar, Saudi Arabia, 2022). URL https://doi.org/10.1109/CICN56167.2022.10008343.

[24] Sadad, N. U., Afrin, A. & Mondal, M. N. I. *Binary classification using k-nearest neighbor algorithm on fpga*, 1–4 (Rajshahi, Bangladesh, 2021). URL https://doi.org/10.1109/IC4ME253898.2021.9768439.

[25] Ranjan, G. S. K., Verma, A. K. & Radhika, S. *K-nearest neighbors and grid search cv based real time fault monitoring system for industries*, 1–5 (Bombay, India, 2019). URL https://doi.org/10.1109/I2CT45611.2019.9033691.

[26] Abu Alfeilat, H. A. *et al.* Effects of distance measure choice on k-nearest neighbor classifier performance: A review. *Big Data* **7**, 221–248 (2019). URL https://doi.org/10.1089/big.2018.0175.

[27] Vieira, J., Duarte, R. P. & Neto, H. C. knn-stuff: knn streaming unit for fpgas. *IEEE Access* **7**, 170864–170877 (2019). URL https://doi.org/10.1109/ACCESS.2019.2955864.

[28] Iswanto, I., Tulus, T. & Sihombing, P. Comparison of distance models on k-nearest neighbor algorithm in stroke disease detection. *ATCSJ* **4**, 63–68 (2021). URL https://doi.org/10.33086/atcsj.v4i1.2097.

[29] George, S. & Sumathi, B. Grid search tuning of hyperparameters in random forest classifier for customer feedback sentiment prediction. *International Journal of Advanced Computer Science and Applications* **11** (2020). URL https://doi.org/10.14569/IJACSA.2020.0110920.

[30] Greche, L., Jazouli, M., Es-Sbai, N., Majda, A. & Zarghili, A. *Comparison between euclidean and manhattan distance measure for facial expressions classification*, 1–4 (Fez, Morocco, 2017). URL https://doi.org/10.1109/WITS.2017.7934618.

[31] Haviluddin, A. & Others *A performance comparison of euclidean, manhattan and minkowski distances in k-means clustering*, 184–188 (Palu, Indonesia, 2020). URL https://doi.org/10.1109/ICSITech49800.2020.9392053.

[32] Karo Karo, I. M., Khosuri, A. & Setiawan, R. *Effects of distance measurement methods in k-nearest neighbor algorithm to select indonesia smart card recipient*, 209–214 (Bandung, Indonesia, 2021). URL https://doi.org/10.1109/ICoDSA53588.2021.9617476.

[33] Pandit, S. & Gupta, S. A comparative study on distance measuring approaches for clustering. *International Journal of Research in Computer Science* **2**, 29–31 (2011). URL https://doi.org/10.7815/ijorcs.21.2011.011.

[34] Goel, K., Dwivedi, P. & Sharma, O. *Performance analysis of various sorting algorithms: Comparison and optimization*, 1–5 (Dehradun, India, 2023). URL https://doi.org/10.1109/ISED59382.2023.10444609.

[35] Marcellino, M., Pratama, D. W., Suntiarko, S. S. & Margi, K. *Comparative of advanced sorting algorithms (quick sort, heap sort, merge sort, intro sort, radix sort) based on time and memory usage*, 154–160 (Jakarta, Indonesia, 2021). URL https://doi.org/10.1109/ICCSAI53272.2021.9609715.

[36] Pedregosa, F. *et al.* Scikit-learn: Machine learning in python. *Journal of Machine Learning Research* **12**, 2825–2830 (2011). URL http://jmlr.org/papers/v12/pedregosa11a.html.

[37] of California, U. Centers for disease control and prevention, cdc diabetes health indicators dataset (2025). URL https://doi.org/10.24432/C53919.

[38] Sari, M. & Al Maki, W. F. *Improving k-nearest neighbor performance in footwear classification using leave one out cross validation* (2023). URL https://doi.org/10.1109/ICICyTA60173.2023.10428849.

[39] Sammut, C. & Webb, G. I. (eds). *Leave-One-Out Cross-Validation*, 600–601 (Springer US, Boston, MA, 2010). URL https://doi.org/10.1007/978-0-387-30164-8_469.