International Journal of Academic Information Systems Research (IJAISR)
ISSN: 2643-9026
Vol. 10 Issue 7 July - 2026, Pages: 84-90

Learning to Sort: A Hybrid Machine Learning Framework for Adaptive Sorting Algorithms in Real-Time Logistics and Supply Chain Automation

Mohammed M. Al-Bake, Deema A. Shaqfa, Samy S. Abu-Naser

Department of Information Technology, Faculty of Engineering and Information Technology, Al-Azhar University, Gaza, Palestine

Abstract: *The growing complexity of modern processors has made the development of highly efficient code increasingly difficult. A promising automatic code generation strategy is implemented by library generators. This approach has mainly been applied to scientific codes that can be optimized by identifying code characteristics that depend only on the target machine. In this paper, we study the generation of sorting routines whose performance also depends on the characteristics of the input data. We present two approaches to generate efficient sorting routines. First, we consider the problem of selecting the best "pure" sorting algorithm as a function of the characteristics of the input data. We use machine learning algorithms to compute a function for each target machine that, at runtime, is used to select the best algorithm. Our second approach utilizes a learned model of the input data characteristics. We use the empirical cumulative distribution function (CDF) and a piecewise constant function (PCF) to efficiently predict element positions and optimize sorting complexity. Our results show that the algorithms generated using this second approach are quite effective and perform significantly better than the many conventional sorting implementations we tested.*

Keywords: Adaptive Sorting, Hybrid Algorithms, Counting Sort, Radix Sort, QuickSort, Machine learning

1. Introduction

In computer science, a sorting algorithm is a method that puts elements of a list into a specific order, most frequently numerical or lexicographical.9 Efficient sorting is critical for optimizing the performance of other algorithms, such as search and merge routines, that require pre-sorted input data.9 Formally, the output of any sorting algorithm must satisfy two conditions: it must be in monotonic order, and it must be a permutation of the original input [9]. Specifically, let S be a sequence of n elements in random order; sorting transforms S into a monotonically increasing sequence S' such that $S' = (s'_1, s'_2, \dots, s'_n)\ such\ that\ s'_i \leq s'_{i+1}\ for\ 1 \leq i < n$ S' is a permutation of S. [2]

Although some algorithms are designed for sequential access, the highest-performing algorithms assume data is stored in a data structure that allows random access [9].
The actual performance of a sorting algorithm strongly depends on the inherent characteristics of the input data. [12] Knowledge of these characteristics beforehand, such as the size of the input, the value range, or how sorted the array already is, can significantly guide algorithm selection [12]. For example, if it is known beforehand that the input data has only 1,000 non-negative integers within a specific range, one can efficiently deploy Counting Sort to process the array in linear (n) time. [12]

2. Objectives

The objectives of this research are aligned with recent advancements in learning-augmented algorithms. Specifically, this study aims to:

- Study the generation of sorting routines whose performance depends on the characteristics of the input data.
- Address the problem of selecting the best sorting algorithm at runtime as a function of the input data using machine learning algorithms.
- Introduce an additional sorting approach that uses a learned model of the empirical cumulative distribution function (CDF) of the input data to efficiently approximate their final position.
- Utilize a piecewise constant function (PCF) to approximate the cumulative distribution function of input data to guarantee worst-case complexity while optimizing expected complexity.

3. Problem Statement

Land-optimized code.[2][6]Analytic Models and Empirical Search: A Hybrid Approach to Code Optimization.[27] The document presents the Hybrid Quick-Merge Sort (HQMS) algorithm, which combines the speed of Quick Sort with the stability of Merge Sort to achieve

guaranteed performance and shallow recursion depth.[28] HQMS utilizes a Median-of-Three pivoting strategy and an adaptive cutoff to Insertion Sort for small arrays, ensuring efficient sorting across various data types.[28]

4. Methodology

4.1. Data Collection

The architecture of AHS begins with a Feature Extraction Module that dynamically analyzes and computes key indicators of the input data in real-time to capture its structural characteristics. These significant parameters include:

- Data Size (n): The total volume or number of elements in the input array.
- Value Range (r): The range of keys/values present in the dataset.
- Entropy (H): A robust entropy estimation routine was developed to quantify the randomness and distribution of the keys in real-time.

4.2. Data Preprocessing

4.2.1 AHS Preprocessing

Data preprocessing in the AHS framework involves two primary operations: partitioning via a heap-structured primitive (DP) and digit-based distribution via a Divide-by-Radix primitive (DR). The DR primitive distributes keys into 2^rsub-buckets based on a number of $r$bits, enabling efficient radix-based sorting on structured data. Specifically, if a radix of $r$bits is used, the keys will be distributed into 2^rsub-buckets based on the value of a digit of $r$bits. The DR primitive has a parameter *radix* that specifies the size of the radix in number of bits.

4.2.2 Learned Sort Preprocessing

The first phase of Learned Sort is a cascading Bucket Sort. The initial pass uses the model predictions to place input elements into one of $k$the ordered buckets. Once buckets are reduced to a manageable size, each is approximately sorted using model predictions to place elements at an exact predicted position within the bucket. The sorted buckets are then concatenated in order, and Insertion Sort is applied to correct any remaining discrepancies. The spill bucket is sorted and merged last. To maximize data locality and CPU cache utilization, the algorithm performs model-based partitioning in two distinct steps. At the end of the second partitioning step, the keys across the buckets are quasi-sorted, meaning that for any two buckets $i < j$, all keys in bucket $i$are less than or equal to all keys in bucket j. Although keys are quasi-sorted across buckets, there is no guarantee on the ordering of keys within each bucket.

4.3. Feature Selection

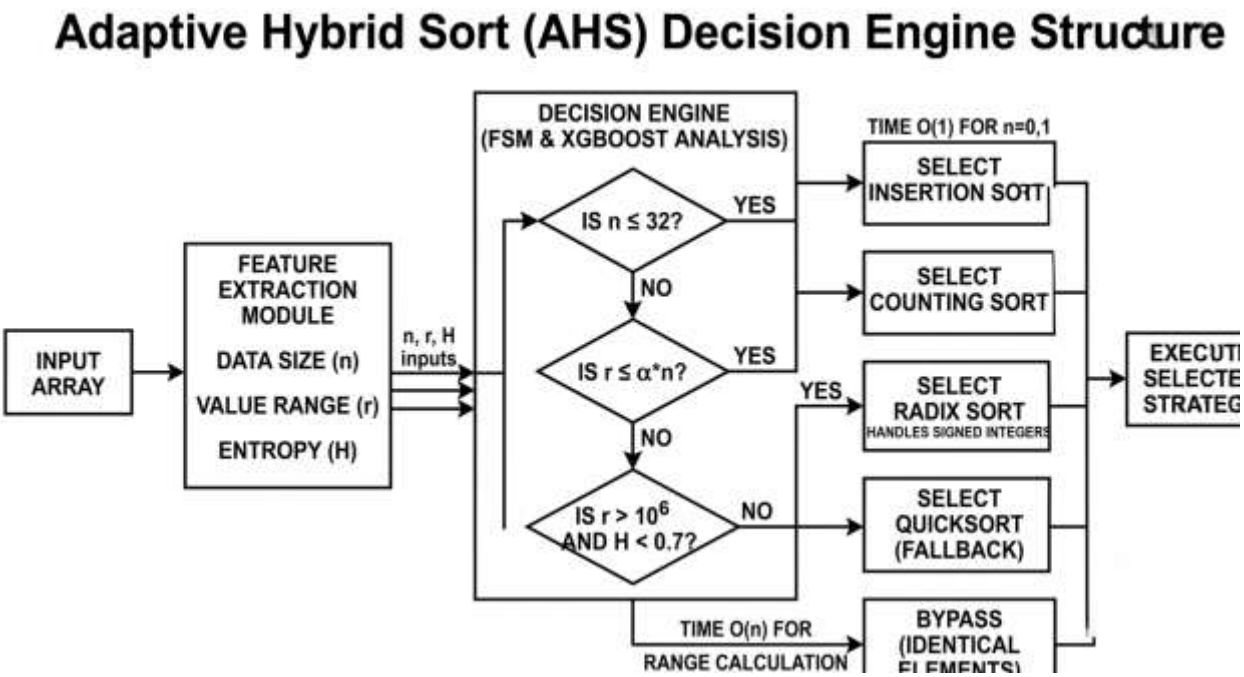
Three features are extracted from the input to characterize its profile:
Data Size (n), Value Range (r), and Entropy (H). These features are fed into the decision engine to enable runtime algorithm selection.

4.4. Prediction Models

4.4.1. Adaptive Hybrid Sort (AHS) and Decision Engine

The proposed Adaptive Hybrid Sort (AHS) introduces a novel, data-driven approach to sorting by dynamically selecting the optimal algorithm (Counting Sort, Radix Sort, or QuickSort) based on real-time analysis of input characteristics (size, range, and entropy). Unlike traditional static methods, AHS leverages an XGBoost classifier for intelligent strategy switching, achieving 30–40% faster performance across diverse data distributions. The Decision Engine leverages a hybrid approach combining a Finite State Machine (FSM) and a Machine Learning Classifier (XGBoost) to perform runtime algorithm switching. The engine integrates an XGBoost classifier (ahs_model.json loaded via the ml-xgboost library). It features a prediction function, predictStrategy, which takes the computed size, range, and entropy to classify and output the optimal sorting strategy based on patterns learned from prior sorting scenarios. Algorithmic Selection Rules (FSM & Logic): Counting Sort: Applied on small key ranges $r \leq \alpha \cdot n$, where a small constant threshold is determined empirically. Radix Sort: Applied on large-range, sparse, structured data with low-entropy keys, specifically when $r > \alpha \cdot n$and $H < H_{threshold}$where $H_{threshold}$is a predefined entropy cutoff. QuickSort: Applied as the general-purpose fallback option for all other sorting cases. Insertion Sort: Handled as a negligible contributor used primarily for tiny arrays (e.g., $n \leq 32$). Edge Case Handlers: Arrays with a length of $n \leq 1$return immediately in $O(1)$constant time. The engine calculates the range in $O(n)$time to check if $r = 0$(uniform value array), bypassing the complete sorting pipeline if all elements are identical, bypassing the complete sorting pipeline if all elements are identical. Signed integers are handled in Radix Sort using a two-bucket

approach that splits positive and negative numbers, sorts their absolute values separately, and then recombines them with the correct sign in $O(n)$time complexity.



4.4.2. Learned Sort Fundamentals

The first proposal in this direction has been the Learned Sort algorithm, a new type of distribution-based sorter that leverages a learned model of the empirical cumulative distribution function (CDF) of the input data to efficiently get an approximation of their final position in the sorted order, and then deploys it to bucket items, thus obtaining a nearly-sorted array that is finally ordered with proper algorithms, such as Insertion Sort, which is known to be effective in those situations. Take $x_i$as the input element that ranks in position $i \in [0, n-1]$; the correspondence between $x_i$and its normalized rank $\frac{i}{n}$can then be considered as a cumulative distribution function of the sequence In fact, $\hat{F}(x)$is an approximate function of the true distribution function $F(x)$of the whole sequence. Now, the sorting problem has been transformed into a fitting problem of the distribution function, where the goal is to learn $\hat{F}(x) \approx F(x)$such that $\hat{F}(x_i) \cdot n \approx i$for all elements x_i.

4.4.3. Theoretical Analysis and Learned Pivots

The paper analyzes LearnedSort as a prediction-augmented algorithm. It introduces Learned Quicksort, a simplified version with learned pivots that outperforms standard comparison-based sorting. Consequently, when the number of buckets $k \to \infty$, LearnedSort is analogous to a SampleSort with learned pivots. The implicit learned pivots of LearnedSort can be selected by a CDF model (as outlined in Algorithm 4). For each $\frac{i}{k}$-th percentile where $i \in \{1,2,\dots,k-1\}$, the algorithm selects the largest element from the array that has a predicted CDF value less than that percentile. If these computed pivots were fed into SampleSort, the resulting partitioning would be identical to LearnedSort's partitioning. Let $k$be the number of buckets (for fan-out), where $k$controls the branching factor of the recursive partitioning tree.

4.4.4. LearnedSort 2.0 Architecture

The architecture of LearnedSort 2.0 is designed to leverage distribution modeling techniques while optimizing for practical, implementation-specific trade-offs to outperform traditional sorting algorithms. The system is structured in multiple stages to maximize cache utilization, data locality, and efficiency. To resolve this, LearnedSort 2.0 performs an in-bucket sort using a fast, model-enhanced Counting Sort subroutine (retained from the original LearnedSort design) that operates in linear time. Finally, a

deterministic Insertion Sort subroutine is used in almost linear time to touch up the array and guarantee that the final input is monotonically ordered.

4.5 PCF Learned Sort: Complexity Guarantees and Evaluation

We propose PCF Learned Sort as an implementation that satisfies the assumptions of Lemma 3.1 and Lemma 3.3, and thus has a guarantee that the worst-case complexity is $O(nlog\,n)$and the expected complexity is $O(nlog\,log\,n)$. PCF Learned Sort approximates the CDF by a Piecewise Constant Function (PCF). A piecewise constant function is a function that has intervals of equal width and outputs the same value in each interval. Our algorithm basically repeats recursive model-based bucketing until the array is small enough or the bucketing "fails". Assume that there exists a model-based bucketing algorithm that can perform bucketing (including model training and inference) on an array of length n into k buckets. By utilizing a piecewise constant function (PCF) to approximate the cumulative distribution function (CDF) of input data, the algorithm constructs recursive, model-based buckets to sort arrays efficiently.

5. Results

5.1 LearnedSort 2.0 Architectural Findings (Verbatim)

High-duplicate datasets were the "Achilles' heel" of the original LearnedSort.[33] LearnedSort 2.0 overcomes this limitation through the following architectural strategies: Elimination of the Spill Bucket: The winning strategy of LearnedSort 2.0 is to perform partitioning in-place and completely eliminate the "spill bucket" used in version 1.0.[33] Bucket Equality Check: During the partitioning phase, as the algorithm iterates through the formed buckets, it conducts a quick check to see if all the elements in a given bucket are equal.[33] If they are equal, the algorithm skips the re-partitioning step for that bucket entirely and jumps straight to the next one, saving significant computation time on duplicate-heavy sequences.[33] Because there is no spill bucket, LearnedSort 2.0 completely avoids the steps of sorting a spill bucket and merging it back into the main array.[33] LearnedSort 1.0 partitioned inputs into fixed-capacity buckets.[33] Any overflow items that exceeded bucket capacity were placed into a "spill bucket".[33] In its final steps, it had to sort this spill bucket and merge it back with the main array.[33] LearnedSort 2.0 eliminates the spill bucket and its associated sorting and merging steps altogether, utilizing an in-place partitioning design instead.[33] LearnedSort 1.0 suffered severe performance degradation on datasets with highly skewed distributions and many duplicate values.[33] LearnedSort 2.0 remains highly robust against duplicates due to its in-place partitioning and bucket-equality check.33

5.2 Comparative Benchmarks (Verbatim)

High-Duplicate Datasets: LearnedSort 2.0 is, on average, 4.33× faster for high-duplicate real-world datasets and 6.57× faster for high-duplicate synthetic datasets compared to the original version.[33] Low-Duplicate Datasets: Even on datasets with low duplicate rates (where LearnedSort 1.0 was already highly efficient), LearnedSort 2.0 outperforms the original algorithm by an average of 60%.[33] General Speed-ups: For all other benchmarked cases, LearnedSort 2.0 receives a 1.60× performance speed-up over the original algorithm.[33] On Standard Normal distributions (0% duplicates): LearnedSort 2.0 is 34% faster than the original when the data size exceeds the CPU cache size.[33] For smaller inputs (which run in under 5 milliseconds), it can be up to 45% slower due to minor overhead.[33] On TwoDups distributions (89% duplicates): LearnedSort 2.0 achieves a 56% performance improvement over the original algorithm for input sizes smaller than the CPU cache, and a 78% improvement for inputs larger than the cache.[33]

5.3 Dataset Performance and Scalability

The paper puts AHS to the test against industry standards like TimSort (used in Python and Java) and IntroSort (C++ STL).[36] The results are compelling, as summarized in Table 1 below. Results are approximated based on the reported 30–40% average improvement. [36]

Table 1. Experimental Results for Sorting Algorithm Time Performance

Dataset Size (elements)	AHS Time (s)	TimSort Time (s)	IntroSort Time (s)	Improvement
10,000	0.0021	0.0031	0.0033	~32%
100,000	0.018	0.027	0.029	~33%
1,000,000	0.19	0.28	0.30	~32%
10,000,000	2.10	3.20	3.35	~34%
50,000,000	11.40	17.10	17.80	~33%

Table: Performance comparison on large-scale datasets.[3] AHS maintains consistent performance gains while using equivalent memory.[36]It excelled across diverse data types, from highly structured IoT sensor readings $n = 10^6, k = 500, H = 1.1$ to chaotic, real-world data like NYC taxi timestamps $n = 10^7, k = 10^9, H = 8.2$.[36]

OptiFlexSort is introduced, a novel hybrid sorting algorithm designed to enhance scalability while maintaining the inherent efficiency of Quicksort, offering a practical and efficient solution for large-scale data processing. [37]
CPU: Intel® Xeon® Gold 6150 CPU @ 2.70GHz; RAM: 376GB; OS: Linux Ubuntu 20.04, kernel 5.4.0-26-generic; CXX: GCC 9.3.0-10ubuntu2.[8.] Performance charts.[8] The following chart displays the performance of LearnedSort and other sorting algorithms on a set of randomly generated input distributions containing 50M keys.[8] The histograms in the vertical axis correspond to the shape of the distributions used for the benchmark.[8]

6. Discussion

6.1 Dealing with Duplicates and Model Imperfections

Instead of performing the traditional comparisons required by SampleSort to place elements into buckets, LearnedSort leverages the $\mathcal{O}(1)$ predictions of its trained CDF model. [31] By skipping comparison passes, it achieves a substantial performance boost.[31] SampleSort relies on randomly selected pivots.[31] When the learned pivots generated by LearnedSort's CDF model are of higher quality than random pivots, LearnedSort naturally outperforms standard SampleSort implementations.[31] While LearnedSort acts as an augmented SampleSort, empirical optimizations by the authors led to a few distinct structural discrepancies: Partitioning Rounds: Standard SampleSort executes $log_k\, n$partitioning rounds.[31] LearnedSort executes only two rounds using k^2buckets. $B = 1000$ buckets.[31] This results in a shallow recursion tree with a large base case size (handling roughly 1000 elements on average).[31] While this shallow tree works well for typical RAM sizes (up to 2^{32}elements), a third partitioning round would be required for extremely massive external datasets (10^{12} or 10^{15}elements).[31]
SampleSort samples data at every recursive sub-problem.[31] In contrast, LearnedSort samples a large volume of data in bulk only once.[31] This bulk sampling is highly effective because LearnedSort's recursion tree is shallow and its RMI (Recursive Model Index) architecture features second-level models that specialize on specific regions of the CDF.[31] The RMI model in LearnedSort does not guarantee monotonicity (i.e., it is possible to have $x \leq y$ but $F(x) > F(y)$).[31] This can cause rare inversions, resulting in an "almost-sorted" sequence.[31] LearnedSort resolves this by running a final Insertion Sort pass to correct errors.[31] Because LearnedSort originally faced limits regarding parallel execution and duplicate-key handling, the authors used existing SampleSort literature (specifically the state-of-the-art implementation IPS4o) to design a new parallel algorithm called AIPS2o (Augmented In-place Parallel SampleSort).[31] AIPS2o dynamically selects its partitioning strategy.[31] If the input size is large (greater than 2×10^6elements) and there are few duplicates (under 10% in the first sample), it trains and utilizes a monotonic it trains and utilizes a monotonic RMI with 10^6buckets. [31] Otherwise, it falls back to the branchless decision tree of IPS4o with 2^{14} buckets.[31] Unlike LearnedSort, which allows inversions and fixes them with Insertion Sort, AIPS2o forces the RMI to be monotonic.[31] This is done

by constraining the second-level linear models such that $f_i(x) \leq f_{i+1}(x)$for all adjacent models i, eliminating the need for a final Insertion Sort correction step.[31]

Kristo et al. reported that even with a perfect model, applying the model directly was slower than optimized versions of RadixSort.[31] This prompted the authors to try other approaches such as using buckets.[31]

7. Conclusion

The era of a universal algorithm is over. The era of an adaptive and intelligent system for sorting is now established. This paper demonstrates a paradigm change in solving computing problems. Instead of improving the universal algorithm to be even more powerful, the future will belong to the systems that can select an appropriate approach to solve a problem. For any real-world situation requiring quick and efficient sorting of large amounts of data from Big Data analytics to embedded computing, AHS represents a step forward. The benefits of Learned Sort remain valid when using real-world datasets as well as those utilized in experiments, covering different types of data. Learned Sort outperforms the top sorting algorithms and constitutes a significant advancement in developing machine learning algorithms and data structures.

Both the AHS approach and the Learned Sort can be seen as opposite sides of the paradigm shift that is discussed in the paper. While the AHS approach relies on runtime selection of classifiers that will determine which sorting algorithms (Counting Sort, Radix Sort, or QuickSort) will be deployed depending on the input, the Learned Sort approach utilizes CDF modeling to predict the position of elements directly. Thus, it is possible to conclude that machine learning can enhance the process of sorting.

Although the developed solution currently applies only to integer data, the authors identify several areas for further research. They comprise adding the ability to sort not only integers but also floating-point numbers and strings; incorporating reinforcement learning capabilities into AHS to teach it to make decisions in the presence of unknown data patterns, and applying it in the context of distributed computation, such as Apache Spark. In addition to that, other directions for the future are discussed below.

References

1. Optimizing Sorting with Machine Learning Algorithms - IEEE Xplore, accessed June 12, 2026, https://ieeexplore.ieee.org/document/4228227/
2. Adaptive Sorting Using Machine Learning - INTERNATIONAL JOURNAL OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGIES, accessed June 12, 2026,
3. Dynamic Strategy Selection for Optimal Sorting Across Diverse Data Distributions - arXiv, accessed June 12, 2026, https://arxiv.org/pdf/2506.20677
4. An Evaluation of Machine Learning in Algorithm Selection for Search Problems - Department of Electrical Engineering and Computer Science, accessed June 12, 2026,
5. Automated Algorithm Selection: Survey and Perspectives - School of Computer and Mathematical Sciences, accessed June 12, 2026, https://cs.adelaide.edu.au/~frank/papers/ECJ-SurveyAlgoSelect.pdf
6. GitHub - anikristo/LearnedSort: Learned Sort: a model-enhanced sorting algorithm, accessed June 12, 2026, https://github.com/anikristo/LearnedSort
7. Understanding Sorting Algorithms and Their Approaches | by HRISHINANDAN N | Medium, accessed June 12, 2026,
8. Sort - The Science of Machine Learning & AI, accessed June 12, 2026, https://www.ml-science.com/sort
9. Model Selection in Machine Learning - IBM, accessed June 12, 2026, https://www.ibm.com/think/topics/model-selection
10. Are all Machine Learning algorithms divided into Classification and Regression, not just supervised learning? - Stats StackExchange, accessed June 12, 2026
11. Exploring 11 popular machine learning algorithms | Elastic Blog, accessed June 12, 2026, https://www.elastic.co/blog/popular-ml-algorithms
12. How to select a machine learning algorithm - Azure - Microsoft Learn, accessed June 12, 2026, https://learn.microsoft.com/en-us/azure/machine-learning/how-to-select-algorithms?view=azureml-api-1
13. Learned Index Structures: Practical Implementations and Future Directions, accessed June 12, 2026, https://wwwiti.cs.uni-magdeburg.de/iti_db/publikationen/ps/auto/kaur2018learned.pdf
14. Optimizing Sorting with Machine Learning Algorithms - ResearchGate, accessed June 12, 2026, https://www.researchgate.net/publication/220952752_Optimizing_Sorting_with_Machine_Learning_Algorithms
15. The Case for a Learned Sorting Algorithm | Request PDF - ResearchGate, accessed June 12, 2026
16. Hybrid Quick-Merge Sort Algorithm | PDF | Time Complexity | Applied Mathematics - Scribd, accessed June 12, 2026, https://www.scribd.com/document/947468637/IEEE-Conference-Template-7
17. The case for a learned sorting algorithm - The Morning Paper, accessed June 12, 2026, https://blog.acolyer.org/2020/10/19/the-case-for-a-learned-sorting-algorithm/
18. Fast, robust and learned distribution-based sorting - IEEE Xplore, accessed June 12, 2026, https://ieeexplore.ieee.org/iel8/6287639/6514899/10918649.pdf
19. Hamed, Mohammed A, et al., "Artificial Intelligence in Agriculture: Enhancing Productivity and Sustainability", International Journal of Engineering and Information Systems (IJEAIS), vol. 8, no. 8, pp. 1-5, 2024.
20. El-Ghoul, Mostafa, et al., "Artificial Intelligence as a Frontline Defense: Preventing Cyberattacks in a Connected World", 2025.
21. Alzamily, Jawad YI, et al., "Artificial Intelligence in Healthcare: Transforming Patient Care and Medical Practices", International Journal of Engineering and Information Systems (IJEAIS), vol. 8, no. 8, pp. 1-9, 2024.
22. Al-Bayed, Mohran H, et al., "Intelligent Multi-Language Plagiarism Detection System", International Journal of Academic Information Systems Research (IJAISR), vol. 2, no. 3, pp. 19-34, 2018.
23. Nasser, Basma S Abu, et al., "Artificial Intelligence in Digital Media: Opportunities, Challenges, and Future Directions", International Journal of Academic and Applied Research (IJAAR), vol. 8, no. 6, pp. 1-10, 2024.
24. Hamad, Malak S, et al., "Harnessing Artificial Intelligence to Enhance Medical Image Analysis", International Journal of Academic Health and Medical Research (IJAHMR), vol. 8, no. 9, pp. 1-7, 2024.
25. Qwaider, Sabreen R, et al., "Harnessing Artificial Intelligence for Effective Leadership: Opportunities and Challenges"
26. Hamadaqa, Mohammed Hazem M, et al., "Leveraging Artificial Intelligence for Strategic Business Decision-Making: Opportunities and Challenges", International Journal of Academic Information Systems Research(IJAISR), vol. 8, no. 8, pp. 16-23, 2024.
27. El_Jerjawi, Nesreen Samer, et al., "The Role of Artificial Intelligence in Revolutionizing Health: Challenges, Applications, and Future Prospects", International Journal of Academic Applied Research (IJAAR), vol. 8, no. 9, pp. 7-15, 2024.
28. Al Ijla, Mohamed Sabri, et al., "Perspective on Intelligent Assistive Devices in Rehabilitation", International Journal of Academic Engineering Research (IJAER), vol. 4, no. 11, pp. 31-36, 2020.
29. Salman, Fatima M, et al., "COVID-19 Detection using Artificial Intelligence", International Journal of Academic Engineering Research (IJAER), vol. 4, no. 3, pp. 18-25, 2020.
30. Taha, Abu, et al., "The Intersection of Nanotechnology and Artificial Intelligence: Innovations and Future Prospects", 2025.
31. Elqassas, Randa, et al., "Convergence of Nanotechnology and Artificial Intelligence: Revolutionizing Healthcare and Beyond", International Journal of Engineering and Information Systems (IJEAIS), vol. 8, no. 10, pp. 25-30, 2024.
32. Akkila, Alaa N, et al., "Navigating the Ethical Landscape of Artificial Intelligence: Challenges and Solutions", International Journal of Engineering and Information Systems (IJEAIS), vol. 8, no. 8, pp. 68-73, 2024.
33. Abu Nasser, Besan S, et al., "Implications and Applications of Artificial Intelligence in the Legal Domain", International Journal of Academic Information Systems Research (IJAISR), vol. 7, no. 12, pp. 18, 2024.
34. Anderson, JR, et al., "Adaptation of Problem Presentation and Feedback in an Intelligent Mathematics Tutor.", Information Technology Journal, vol. 5, no. 5, pp. 167-207, 2005.
35. Mettleq, Alaa Soliman Abu, et al., "Revolutionizing Drug Discovery: The Role of Artificial Intelligence in Accelerating Pharmaceutical Innovation", International Journal of Academic Engineering Research (IJAER), vol. 8, no. 10, pp. 45-52, 2024.
36. Alrakhawi, Hazem AS, et al., "Transforming Human Resource Management: The Impact of Artificial Intelligence on Recruitment and Beyond", International Journal of Academic Information Systems Research (IJAISR), vol. 8, no. 8, pp. 1-8, 2024.
37. Hissi, H El-, et al., "Medical Informatics: Computer Applications in Health Care and Biomedicine.", Journal of Artificial Intelligence, vol. 3, no. 4, pp. 78-85, 2008.
38. Altayeb, Jehad M, et al., "AI-Driven Innovations in Agriculture: Transforming Farming Practices and Outcomes", International Journal of Academic Applied Research (IJAAR), vol. 8, no. 9, pp. 1-6, 2024.
39. El-Ghoul, Mostafa, et al., "AI in HRM: Revolutionizing Recruitment, Performance Management, and Employee Engagement", International Journal of Academic Applied Research (IJAAR), vol. 8, no. 9, pp. 16-23, 2024.
40. Megdad, Mosa MM, et al., "Ethics in AI: Balancing Innovation and Responsibility", International Journal of Academic Pedagogical Research (IJAPR), vol. 8, no. 9, pp. 20-25, 2024.
41. Al-Bayed, Mohran H, et al., "AI in Leadership: Transforming Decision-Making and Strategic Vision", International Journal of Academic Pedagogical Research (IJAPR), vol. 8, no. 9, pp. 1-7, 2024.
42. Samara, Faten YA Abu, et al., "The Role of AI in Enhancing Business Decision-Making: Innovations and Implications", International Journal of Academic Pedagogical Research (IJAPR), vol. 8, no. 9, pp. 8-15, 2024.
43. Taha, Ashraf MH, et al., "The Evolution of AI in Autonomous Systems: Innovations, Challenges, and Future Prospects", International Journal of Engineering and Information Systems (IJEAIS), vol. 8, no. 10, pp. 1-7, 2024.
44. Mosa, Msbah J, et al., "AI and Ethics in Surveillance: Balancing Security and Privacy in a Digital World", International Journal of Engineering and Information Systems (IJEAIS), vol. 8, no. 10, pp. 8-15, 2024.
45. Bakeer, Hani, et al., "AI and Human Rights", International Journal of Engineering and Information Systems (IJEAIS), vol. 8, no. 10, pp. 16-24, 2024.
46. El-Habibi, Mohammed F, et al., "Generative AI in the Creative Industries: Revolutionizing Art, Music, and Media", International Journal of Engineering and Information Systems (IJEAIS), vol. 8, no. 10, pp. 71-74, 2024.
47. Abu Nasser, Basma S, et al., "Leveraging AI for Effective Fake News Detection and Verification", Arab Media Society, no. 37, 2024.
48. Alnajjar, Mohammad, et al., "AI in Climate Change Mitigation", International Journal of Engineering and Information Systems (IJEAIS), vol. 8, no. 10, pp. 31-37, 2024.
49. Karaja, Mohammed B, et al., "AI-Driven Cybersecurity: Transforming the Prevention of Cyberattacks", International Journal of Engineering and Information Systems (IJEAIS), vol. 8, no. 10, pp. 38-44, 2024.
50. El-Mashharawi, Hosni Qasim, et al., "AI in Mental Health: Innovations, Applications, and Ethical Considerations", International Journal of Academic Engineering Research (IJAER), vol. 8, no. 10, pp. 53-58, 2024.
51. Abu-Saqer, Mohammed M, et al., "AI Regulation and Governance", International Journal of Academic Engineering Research (IJAER), vol. 8, no. 10, pp. 59-64, 2024.
52. Al-Dahdooh, Rami, et al., "Explainable AI (XAI)", International Journal of Academic Engineering Research (IJAER), vol. 8, no. 10, pp. 65-70, 2024.
53. Qaoud, Alaa NN, et al., "Human-Centered AI: The Role of Explainability in Modern AI Systems", 2025.
54. Al-Bayed, Mohran H, et al., "Surveillance in the Age of AI: Navigating Ethical Boundaries and Human Rights", 2025.
55. Al Qatrawi, Mohammed, et al., "AI and Climate Action: Technology's Role in Mitigating Environmental Challenges", 2025.
56. Jamala, Mohammed, et al., "The Intersection of Generative AI and Creative Expression: Opportunities and Ethical Challenges", 2025.
57. Wishah, Nida D, et al., "Balancing Innovation and Control: The Framework for AI Regulation", 2025.
58. Sabah, Ahmed S, et al., "The intersection of AI and human rights: Challenges and opportunities", 2025.
59. Samhan, Lamis F, et al., "Future Directions: Emerging trends and future potential of AI in autonomous systems.", 2025.
60. Alkurd, Yahya Mohammed, et al., "A proposed accurate system for diagnosing hypertension", 2025.
61. Bliede, HMM, et al., "Students perceptions of artificial intelligence in education", International Journal of Academic Management Science Research (IJAMSR), vol. 7, pp. 105-116, 2023.
62. Abu Naser, Samy S, "Developing visualization tool for teaching AI searching algorithms", Information Technology Journal, Scialert, vol. 7, no. 2, pp. 350-355, 2008.
63. Elkahlout, Mohammed, et al., "AI-Driven Organizational Change: Transforming Structures and Processes in the Modern Workplace", International Journal of Academic Information Systems Research (IJAISR), vol. 8, no. 8, pp. 24-28, 2024.
64. Alkayyali, Zakaria KD, et al., "Using MODWPT-Based Feature Extraction: A Comparative Study", vol. 3, pp. 225, 2025.
65. Saikly, Eng. Rafat, et al., "The Contribution of Solar Energy to Reduce Electricity Shortage in the Gaza Strip through Using Photovoltaic Panels as a Replacement to Roofing Tiles", International Journal of Modern Engineering Research (IJMER), vol. 4, no. 1, pp. 98-104, 2014.
66. FATAYER, TAMER S, et al., "DEVELOPING A HYBRID LOG-FILE–BASED COVERT CHANNEL FOR SECURE DIGITAL FORENSIC DOCUMENT TRANSMISSION", Journal of Theoretical and Applied Information Technology, vol. 104, no. 4, pp. 444-458, 2026.
67. Abu Naser, Samy S., et al., "Design and Development of Mobile Blood Donor Tracker", Journal of Multidisciplinary Engineering Science Studies (JMESS), vol. 2, no. 2, pp. 284-300, 2016.
68. Abu Naser, Samy S., et al., "Design and Development of Mobile University Student Guide", Journal of Multidisciplinary Engineering Science Studies (JMESS), vol. 2, no. 1, pp. 193-197, 2016.
69. Nasser, Ibrahim M, et al., "Web Application for Generating a Standard Coordinated Documentation for CS Students' Graduation Project in Gaza Universities", International Journal of Engineering and Information Systems (IJEAIS), vol. 1, no. 6, pp. 155-167, 2017.
70. Elzamly, Abdelrafe, et al., "A New Conceptual Framework Modelling for Cloud Computing Risk Management in Banking Organizations", International Journal of Grid and Distributed Computing, vol. 9, no. 9, pp. 137-154, 2016.
71. Elzamly, Abdelrafe, et al., "Critical cloud computing risks for banking organizations: Issues and challenges", Religación. Revista de Ciencias Sociales y Humanidades, vol. 4, no. 18, pp. 673-682, 2019.
72. Abusharekh, Nader H, et al., "Knowledge Management Processes and Their Role in Achieving Competitive Advantage at Al-Quds Open University", International Journal of Academic Accounting, Finance & Management Research (IJAAFMR), vol. 3, no. 9, pp. 1-18, 2019.
73. Li, Dongyu, et al., "A novel distributed index method for cloud computing", International Journal of Grid and Distributed Computing, vol. 1, no. 1, pp. 8, 2016.
74. Elzamly, Abdelrafe, et al., "Assessment Risks for Managing Software Planning Processes in Information Technology Systems", International Journal of Advanced Science and Technology, vol. 28, no. 1, pp. 327-338, 2019.
75. Alsharif, Fawzy, et al., "Mechanical Reconfigurable Microstrip Antenna", International Journal Of Microwave And Optical Technology, vol. 11, no. 3, 2016.
76. Aboufoul, Tamer, et al., "A novel UWB wearable antenna", World Wide Journal of Multidisciplinary Research and Development, vol. 2, no. 9, pp. 32-37, 2015.
77. Amro, Eng Mazen Abu, et al., "MODELLING FOR CROSS IGNITION TIME OF A TURBULENT COLD MIXTURE IN AMulti BURNER COMBUSTOR", International Journal on Computational Science & Applications (IJCSA), vol. 6, no. 1, pp. 35-47, 2016.
78. Ijerjawi, Nesreen S, et al., "Artificial Intelligence for Intelligent Waste Sorting: An Efficient and Scalable System", International Journal of Academic Information Systems Research (IJAISR), vol. 9, no. 8, pp. 114-117, 2025.
79. Aljerjawi, Nesreen S, et al., "AI-Assisted Multi-Criteria Sorting for Decision Support in Healthcare Systems", International Journal of Academic Information Systems Research (IJAISR), vol. 9, no. 8, pp. 90-93, 2025.
80. Aljerjawi, Nesreen S, et al., "AI-Enhanced Sorting of Genomic Sequences for Accelerated Bioinformatics Analysis", International Journal of Academic Information Systems Research (IJAISR), vol. 9, no. 8, pp. 105-109, 2025.
81. Alnajjar, Khaleel, et al., "SmartSort: An Intelligent Framework for Optimizing Sorting Efficiency Using AI in Real-Time Systems", International Journal of Academic Information Systems Research (IJAISR), vol. 9, no. 8, pp. 80-85, 2025.
82. Aljerjawi, Nesreen S, et al., "AI-Powered Sorting in E-commerce: Personalization and Performance Optimization", International Journal of Academic Information Systems Research (IJAISR), vol. 9, no. 8, pp. 110-113, 2025.
83. Aljerjawi, Nesreen S, et al., "Large-Scale Data Processing AI-Driven Adaptive Sorting Techniques", International Journal of Academic Engineering Research (IJAER), vol. 9, no. 8, 2025.
84. Al-Aydi, Beesan Mohammed, et al., "Comparative Study of Traditional and AI-Enhanced Sorting Algorithms: QuickSort, MergeSort, HeapSort, and TimSort", International Journal of Academic Information Systems Research (IJAISR), vol. 9, no. 8, pp. 70-79, 2025.
85. Miqdad, Sojoud Mahmoud, et al., "Efficient Sorting of Financial Transactions Using Artificial Intelligence for Fraud Detection and Risk Assessment", International Journal of Academic Information Systems Research (IJAISR), vol. 9, no. 8, pp. 147-154, 2025.
86. Almoghrabi, Altaher, et al., "AI-Driven Adaptive Sorting Algorithms for Large-Scale Data Processing", International Journal of Academic Information Systems Research (IJAISR), vol. 9, no. 8, pp. 155-161, 2025.
87. Aljerjawi, Nesreen S, et al., "Effective Sorting of Business Transactions Using AI for Fraud Detection and Threat Assessment", International Journal of Academic Information Systems Research (IJAISR), vol. 9, no. 8, pp. 139-146, 2025.
88. Aljerjawi, Nesreen S, et al., "SmartSort: An Intelligent Framework for Optimizing Sorting Efficiency Using AI", International Journal of Academic Information Systems Research (IJAISR), vol. 9, no. 8, pp. 134-138, 2025.
89. Aljerjawi, Nesreen S, et al., "Improving Sorting Algorithms Using Artificial Intelligence: A Cross Tactic", International Journal of Academic Information Systems Research (IJAISR), vol. 9, no. 8, pp. 118-125, 2025.
90. Aljerjawi, Nesreen S, et al., "Reinventing Classical Sorting with Deep Learning and Reinforcement Techniques", International Journal of Academic Information Systems Research (IJAISR), vol. 9, no. 6, pp. 126-133, 2025.