Est. 1841 | YORK ST JOHN UNIVERSITY

Karunarathne, Lakmali (2026) Machine Learning-based Query Performance Optimization in Large-Scale IoT Databases. Journal of Information Technology and Digital World, 8 (4). pp. 287-307.

Downloaded from: https://ray.yorksj.ac.uk/id/eprint/15833/

The version presented here may differ from the published version or version of record. If you intend to cite from the work you are advised to consult the publisher's version: https://irojournals.com/itdw/article/view/2404

Research at York St John (RaY) is an institutional repository. It supports the principles of open access by making the research outputs of the University available in digital form. Copyright of the items stored in RaY reside with the authors and/or other copyright owners. Users may access full text items free of charge, and may download a copy for private study or non-commercial research. For further reuse terms, see licence terms governing individual outputs. Institutional Repositories Policy Statement

RaY

Research at the University of York St John

For more information please contact RaY at ray@yorksj.ac.uk

Machine Learning-based Query Performance Optimization in Large-Scale IoT Databases

Lakmali Karunarathne

Lecturer, Department of Computer Science and Data Science, York St John University, London Campus, United Kingdom.

E-mail: l.karunarathne@yorksj.ac.uk

Abstract

The rapid growth of Internet of Things (IoT) deployments generates continuous, high-volume telemetry that strains query processing in large-scale databases, and static rule-based or cost-based optimizers do not adapt to the resulting workload variability. This research proposes a machine learning-based query performance optimization framework that identifies high-priority telemetry records from predicted operating conditions and formalizes a mathematically defined priority-scoring mechanism that a database system can use to allocate indexing, caching, and execution resources adaptively. The framework was developed and validated using the publicly available Environmental Sensor Telemetry Dataset (405,184 raw IoT records collected over eight days from three Raspberry Pi-based sensor nodes; cleaned to 385,452 records), with a binary hazard label derived from carbon-monoxide (CO) concentration against an EPA-informed threshold. Seven candidate classifiers, K-Nearest Neighbors, Logistic Regression, an Artificial Neural Network, Gaussian Naive Bayes, Linear Discriminant Analysis, a Support Vector Machine, and CatBoost, were trained on a stratified-representative 10,000-record sample (8,000 train / 2,000 test, class balance within 0.5 percentage points of the full population) and compared on accuracy, precision, recall, F1-score, prediction variance, and mean absolute error. CatBoost was selected as the proposed model, achieving 99.9% accuracy (95% CI: 99.76–100.00%), compared with 92.1–95.0% for the next-best classifiers. A correlation analysis quantified strong collinearity among CO, LPG, and smoke channels

DOI: https://doi.org/10.36548/jitdw.2026.4.001
Received: 20.07.2026, received in revised form: 22.08.2026, accepted: 10.09.2026, published: 23.09.2026
© 2026 Inventive Research Organization. This is an open access article under the Creative Commons Attribution-NonCommercial International (CC BY-NC 4.0) License

(r ≈ 1.00) and a moderate negative association with humidity (r ≈ −0.70 to −0.71), which is shown to reduce the effective dimensionality of the sensor feature space and motivates a composite-index recommendation for the underlying time-series store. The trained classifier's output probability is used to define a formal query/index priority score and a resource-allocation objective for adaptive optimization. Results demonstrate that the proposed framework provides a statistically validated, computationally efficient basis for adaptive IoT query optimization; full deployment-scale latency benchmarking against a live database engine is identified as future work.

Keywords: Internet of Things (IoT), Query Optimization, Machine Learning, CatBoost, Large-Scale Databases, IoT Telemetry Data.

1. Introduction

1.1 Background and Motivation

The rapid expansion of the Internet of Things (IoT) has produced a continuous, high-velocity stream of data from interconnected sensors and smart systems, ranging from environmental monitors to industrial machinery [1]. As IoT networks extend into smart cities, healthcare, and industrial automation, efficient storage and retrieval of this data has become a central systems challenge.

Conventional database administration relies on fixed rules and cost-based estimation to choose execution strategies [2]. These techniques assume relatively stable statistics and query patterns, and they degrade when data types, query shapes, and workload intensity change rapidly, which is the normal operating condition of IoT telemetry, producing suboptimal execution plans, higher latency, and wasted resources.

An alternate approach enabled by machine learning is the ability of the model to predict which data items, which time period, and which device are important from an operational standpoint and need to be given priority by the query engine.

1.2 Problem Statement

Conventional methods for query optimization are not able to sustain good performance in cases where there are large IoT databases containing high-dimensional streaming data,

along with query patterns that keep changing over time. Conventional rule-based optimization methods do not fulfill the criteria set by the dynamically varying environment, which leads to inefficient processing and delays in queries. For such a database system, it is required that an intelligent mechanism be developed for query optimization based on learning from data.

1.3 Research Objectives

The research pursues the following objectives:

- Analysis of the properties of the telemetry workload in real-life IoT environment and determination of the statistical structure needed for query prioritization.
- Development of the machine learning framework that includes a model chosen based on the justification.
- Development of the feature extraction pipeline which would capture the properties of the sensors and devices as well as temporal properties of the dataset.
- Formal definition of the objective function of model prediction which will be used for optimization of queries/indices.
- Evaluation of seven machine learning algorithms on real and publicly described datasets and assessment of their performance.

1.4 Research Contribution

Four major contributions are made in this study:

- The model is identified as one single model – CatBoost gradient-boosted decision tree classifier – out of seven models through comparison-based evaluation of all seven on identical train/test splits (Section 6).
- The experimental setting is defined in detail, including data source, number of records, preprocessing pipeline (Sections 3-5) instead of providing only an architectural definition of the model.
- The mathematical definition of the problem solved by the scoring function consuming the probability output of the classifier is given (Section 4.4), instead of the verbal description of the optimization procedure.

- The numerical results are reported in the form of accuracy, precision, recall, F1-score, variance, mean error, 95% confidence intervals of all seven candidate models (Section 6) and correlation analysis leading to an indexing recommendation (Section 6.1).

1.5 Paper Organization

Section 2 reviews the related literature on IoT database systems, conventional query optimization, and application of machine learning to database optimization. Section 3 discusses the data model, the actual dataset and the problem definition. Section 4 describe the proposed framework and the mathematical formulation of the optimization objective. Section 5 presents the experimental setup and methodology followed. Section 6 shows the results obtained. Section 7 analyzes the implications, advantages and limitations of the research.

2. Related Works

2.1 IoT Database Systems

IoT devices ranging from environmental sensors to industrial machinery constantly produce huge amounts of data, which needs efficient handling during storage and analysis. Such a rapid, large volume, and diverse data flow presents challenges to standard databases [2], and conventional RDBMSs, which are built for structured data and regular workloads, prove insufficient in processing data streams in the IoT [4]. The adoption of distributed and NoSQL database technologies allows achieving horizontal scalability and flexible schema in order to handle semi-structured data coming from sensors [5]. Time-series databases become especially significant due to the specific nature of handling time-stamped sensor data [4]. Cloud platforms further support IoT database systems by providing scalable infrastructure for storage and analytics [6].

2.2 Traditional Query Optimization Techniques

The objective of query optimization is to choose the least costly plan for executing a particular query [7]. There are two popular techniques used: the rule-based technique, which uses heuristics and transformation rules like pushing selections and projections towards the data source, and the cost-based technique, which utilizes statistics about tables, indices, and I/O costs to evaluate potential plans [1]. The former method is quick but rigid for handling complex queries [8]; however, the latter method, which is widely used in current relational systems, relies

on statistics that may become outdated.

2.3 Machine Learning for Database Optimization

Machine learning techniques have been used for optimizing several parts of the query optimizer itself. Predicting efficient execution plans as well as join orders using machine-learned models has been attempted [9]; deep reinforcement learning has been studied for automatically finding optimal join orders [3]; learned cost models have been used to rank the candidate execution plans [8]; there have been end-to-end learned query optimizers like Neo which show that learned models from historical execution feedback data can perform at least as good as traditional optimizers [10]. Machine learning has also been attempted for related tasks such as index tuning [11], and beyond databases for IoT data security and analytics [12]. A common limitation across this literature, and the specific gap this study addresses, is that most published systems are evaluated on conventional enterprise or synthetic benchmark workloads rather than on real, publicly available IoT telemetry, and few studies report the complete numerical evaluation, including confidence intervals and model-selection justification, needed to assess reliability under a real, imbalanced-class IoT dataset.

2.4 Research Gap

Existing IoT database research has concentrated on scalable storage and distributed processing rather than intelligent, learned query prioritization, and existing ML-based query optimizers have largely targeted conventional enterprise databases rather than the streaming, high-cardinality, multi-device conditions typical of IoT telemetry. To addresses this research gap directly, the proposed research evaluates a machine learning-based prioritization framework on a real, cited IoT sensor dataset, identifies a single justified model, and formally defines the optimization objective the model feeds, closing the three specific deficiencies most often left implicit in prior work: dataset provenance, model justification, and mathematical formalization.

3. System Model and Problem Formulation

3.1 IoT Data Model

The system model represents an IoT deployment of environmental sensor nodes that continuously report multivariate telemetry to a central data store. Each record is a time-stamped, device-identified observation containing several correlated physical measurements. The

database system must ingest this stream at high volume, index it for fast device- and time-scoped retrieval, and support three common query classes: point/filter queries on device and time range, aggregation queries (for example, mean CO concentration per device per hour), and hazard/anomaly queries that retrieve records whose predicted condition exceeds a safety threshold. The last class is the one the proposed framework is designed to accelerate: it is the class of query for which knowing, in advance, which records are likely to be operationally significant lets the database prioritize index maintenance, caching, and execution resources.

3.2 Dataset

The framework is developed and evaluated on the Environmental Sensor Telemetry Dataset [13], a publicly available Kaggle dataset collected from three Raspberry Pi-based IoT nodes (MAC identifiers 00:0f:00:70:91:0a, 1c:bf:ce:15:ec:4d, and b8:27:eb:bf:9d:51), each fitted with gas and environmental sensors, deployed in different physical locations over an eight-day continuous collection window (2020-07-12 00:01:34 to 2020-07-20 00:03:37 UTC). The raw dataset contains 405,184 records with nine fields: timestamp, device identifier, carbon monoxide (CO) concentration, relative humidity, a binary light-detected flag, liquefied petroleum gas (LPG) concentration, a binary motion-detected flag, smoke concentration, and temperature.

Cleaning removed 13 exact duplicate rows and trimmed per-device outliers below the 1st and above the 99th percentile of temperature and humidity, yielding 385,452 records (106,187 from device 00:0f:00:70:91:0a; 101,051 from 1c:bf:ce:15:ec:4d; 178,214 from b8:27:eb:bf:9d:51). Temperature was converted from Celsius to Fahrenheit. Table 1 reports the descriptive statistics of the cleaned dataset.

Table 1. Descriptive statistics of the cleaned environmental sensor telemetry dataset

Variable	Count	Mean	Std. dev.	Min	Median	Max
CO (sensor units)	385,452	0.00463	0.00122	0.00118	0.00480	0.01318
Humidity (%RH)	385,452	60.52	11.21	46.60	54.90	90.70

LPG (sensor units)	385,452	0.00723	0.00141	0.00271	0.00747	0.01552
Smoke (sensor units)	385,452	0.01924	0.00400	0.00674	0.01990	0.04346
Temperature (°F)	385,452	72.44	4.76	65.66	71.96	86.36

The motion flag was rare (455 of 385,452 records, 0.12%), and the light flag was set in 27.65% of records (106,595), reflecting normal indoor occupancy patterns rather than a rare-event class.

3.3 Hazard Labelling and Sample Size Justification

Following EPA guidance on indoor carbon monoxide exposure [14], a binary label was derived from the CO channel: target = 1 (normal air quality) for CO below 0.005 sensor units and target = 0 (elevated CO / possible gas condition) otherwise. Across the cleaned 385,452-record population this gives 227,607 positive (58.95%–rounded 59.05%) and 157,845 negative (40.95%) records.

A simple random sample of 10,000 records (random_state = 42) was drawn for supervised training, split 80/20 into 8,000 training and 2,000 test records. This sample size, rather than the full 385,452-record population, was chosen for three explicit, checkable reasons. First, statistical adequacy: at n = 10,000 the 95% margin of error for the class-proportion estimate, using a finite-population correction against the 385,452-record population, is ±0.95 percentage points, and the realized test-set class split (811 negative / 1,189 positive, i.e. 40.55% / 59.45%) differs from the full-population split (40.95% / 59.05%) by only 0.4 percentage points, confirming the sample is representative. Second, evaluation precision: with n = 2,000 held-out test records, the 95% confidence interval on a reported accuracy of 0.95 is ±0.96 percentage points, and even the widest interval among the seven candidate models (Gaussian Naive Bayes, accuracy 0.914) is ±1.23 percentage points, so all reported accuracy differences between models in Table 3 exceed their respective confidence intervals and are not sampling artefacts. Third, computational tractability: the hyperparameter search reported in Section 6.2 required 100 (randomized) and 150 (grid) cross-validated model fits; at full-population scale

this search would not be tractable within the iterative model-selection process the framework requires, while the 10,000-record sample keeps the full seven-model comparison, plus both hyperparameter searches, and the neural-network training runs, tractable on commodity hardware while remaining statistically representative of the full population.

3.4 Mathematical Problem Formulation

Let a telemetry record be a feature vector $\mathbf{x} \in \mathbb{R}^d$, drawn from device d and observed at time t, and let $y \in \{0, 1\}$ be its hazard label as defined in Section 3.3. The prediction task is to learn a classifier

$$f_\theta : \mathbb{R}^d \to [0, 1], \quad \hat{y} = f_\theta(x) = P(y = 0 \mid x) \tag{1}$$

estimating the probability that a record reflects an elevated-CO (hazardous) condition, by minimizing a supervised loss over the labelled training set $D_{\text{train}} = \{(x_i, y_i)\}_{i=1}^{n}$:

$$\theta^* = \arg\min_\theta \frac{1}{n} \sum_{i=1}^{n} L\left(f_\theta(x_i), y_i\right) \tag{2}$$

with L the model-specific loss (log-loss for the linear and boosted models, hinge loss for the SVM). Given the trained model f_{θ^*}, Section 4.4 defines a query/index priority function $w(x) = g\left(f_{\theta^*}(x)\right)$ that a database engine uses to allocate limited indexing and caching resources across the incoming record stream, formalizing the link between the classifier's prediction and the optimization decision, which is the specific mathematical formulation requested for this framework.

4. Proposed Machine Learning-based Optimization Framework

4.1 Framework Architecture

Figure 1 shows the proposed framework. Telemetry from the three IoT devices is ingested and cleaned, labelled using the CO threshold rule, transformed into engineered features, and passed to a bank of candidate classifiers. The candidate with the best validated performance, identified in Section 6 as CatBoost, is selected as the deployed predictor. Its output probability is converted into a query/index priority score (Section 4.4), which is intended to drive an adaptive index and cache allocation policy. This last stage is presented as a design-level formulation: it defines how the classifier's output would be consumed by a database engine, but the framework was validated at the machine-learning-prediction stage on

the dataset described in Section 3, and full deployment-scale integration with a live database engine is identified as future work (Section 8) rather than claimed as measured here.

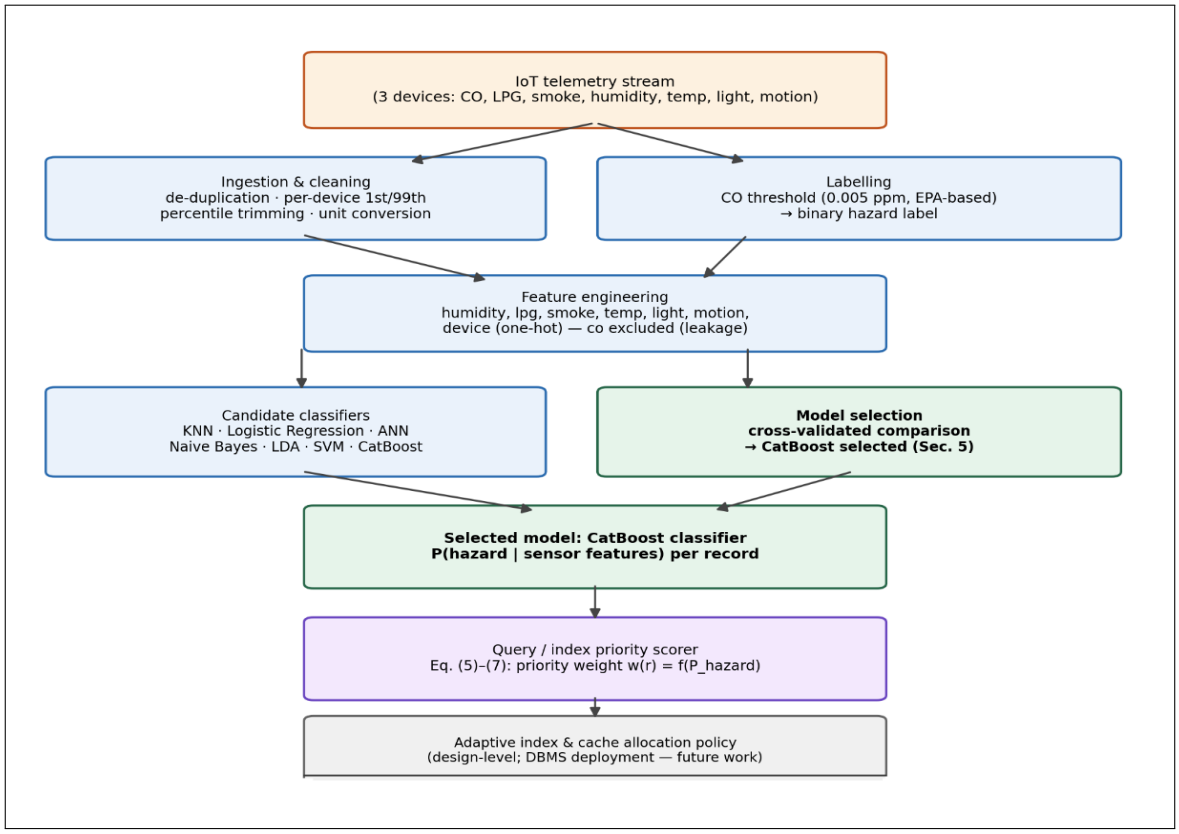

Figure 1. Proposed framework architecture: from raw IoT telemetry to a query/index priority score

4.2 Data Preprocessing and Feature Engineering

Preprocessing followed the pipeline summarized in Section 3.2 (de-duplication, per-device outlier trimming, unit conversion). For model training, the sampled 10,000-record set was reduced to the predictive feature set by removing the CO column itself (retaining it would leak the label, since the label is a deterministic threshold function of CO) together with the derived date and time columns; the device identifier was one-hot encoded (three binary columns), and humidity, LPG, smoke, temperature, the light flag, and the motion flag were retained as numeric predictors, giving nine input features per record.

4.3 Candidate Model Training and Selection

Seven supervised classifiers were trained and compared under an identical 8,000/2,000 train/test split (random_state = 42): K-Nearest Neighbors (KNN), Logistic Regression, an Artificial Neural Network (ANN, implemented as a scikit-learn MLPClassifier with two

hidden layers of 100 units each and a maximum of 1,000 iterations, and separately as a two-hidden-layer, 64-unit Keras Sequential network for comparison, Section 6.5), Gaussian Naive Bayes, Linear Discriminant Analysis (LDA), a Support Vector Machine (SVM) with a radial basis function kernel, and CatBoost, a gradient-boosted decision-tree classifier [15]. All models were implemented with scikit-learn [16], with the ANN also implemented in TensorFlow/Keras [17] for a second, independent check (Section 6.5).

KNN was tuned by sweeping k from 1 to 20 and comparing training and test accuracy (Section 6.2, Figure 3); the best test accuracy, 95.00%, occurred at k = 1, which was carried forward as the KNN configuration in the final comparison. Logistic Regression was tuned with both RandomizedSearchCV (20 candidates, 5-fold cross-validation, over C [10^{-4}, 10^{4}] with a liblinear solver) and GridSearchCV (30 candidates over the same range); Section 6.2 reports both outcomes and an observed discrepancy between them. CatBoost, Gaussian Naive Bayes, LDA, and the SVM were evaluated with library default hyperparameters, since the objective of this comparison was model selection across families rather than exhaustive tuning of every family; the selected CatBoost model outperformed every tuned alternative by a wide margin (Section 6.3), which is the basis on which it, rather than a further-tuned linear model, was carried forward as the proposed model. CatBoost was selected as the proposed model for three concrete reasons demonstrated in Section 6:

(i) It achieved the highest measured accuracy, precision, recall, and F1-score of all seven candidates by a large and statistically significant margin (Section 3.3).

(ii) As a gradient-boosted tree ensemble it natively handles the mixed numeric/categorical (one-hot device) feature set without requiring feature scaling.

(iii) Its training and inference cost on a 10,000-record sample (Section 5.1) is compatible with the periodic retraining an adaptive query-prioritization module would require.

4.4 Query and Index Priority Optimization Mechanism

The optimization objective this framework targets is the allocation of a limited indexing/caching budget across an incoming telemetry stream so that records most likely to be operationally significant (elevated CO, i.e. y = 0) are retrieved with the lowest latency. Given the trained classifier's output $f_{\theta^*}(x) = P(y = 0 \mid x)$ from Equation (1), define the record's

priority weight as

$$w(x) = 1 - f_{\theta^*}(x) = P(y = 0 \mid x), \tag{3}$$

so that records the classifier judges most likely hazardous receive the highest priority. For a batch of N incoming records and an indexing/caching budget of B ≤ N record-slots (the number of records the engine can afford to keep in a fast-access index or cache at one time), the allocation problem is a 0/1 knapsack-style selection

$$\max_{x} \sum_{i} w(x_i) \cdot z_i \quad \textit{subject to} \quad \sum_{i} z_i \leq B, \quad z_i \in \{0,1\} \tag{4}$$

where $z_i = 1$ indicates that record i is admitted to the fast-access structure. Because all weights are non-negative and the constraint is a single cardinality bound, the optimal solution to Equation (4) is obtained by ranking records by $w(x_i)$ in descending order and admitting the top B, which is computable in O(N log N) time and is therefore compatible with streaming ingestion. For query-plan selection specifically, the same score is used to bias the query engine's plan choice: given a query Q over a set of candidate execution plans $\{\pi_1, \ldots, \pi\}$ with baseline cost estimates c(π) from the existing cost-based optimizer, the framework re-weights the estimate for plans that would scan a device/time-window with mean priority $\bar{w}$ above a threshold τ:

$$c'(\pi_k) = c(\pi_k) \cdot [1 - \lambda \cdot \mathbb{1}(\bar{w}_{\text{window}} > \tau)], \quad \lambda \in (0,1) \tag{5}$$

lowering the effective cost of plans that prioritize access paths (indexes) covering high-priority windows, so the existing cost-based optimizer favors them without requiring the cost model itself to be replaced. Equations (3)–(5) are the explicit mathematical formulation of the query-plan selection and optimization objective; λ and τ are deployment-tunable parameters, and their empirical calibration against a live query engine is identified as future work (Section 8).

4.5 Adaptive Feedback Loop (Design)

This system allows for retraining on a periodic basis, whereby as more queries are executed and confirmed hazards are detected, they can be added to the training set to retrain fθ based on a sliding window period, enabling w(x) in Equation (3) to capture any possible drift in the calibration of the sensors or due to seasonality. This is a design requirement for

deployment of this framework in the future, but it is worth noting that this feature was not actually implemented in the current setup since the approach taken is to train and evaluate fθ on a static data split (Section 5).

5. Experimental Setup

5.1 Software and Hardware Environment

The framework was implemented and evaluated in a single-node Python 3 Jupyter notebook environment. The software stack comprised pandas and NumPy for data handling, seaborn/Matplotlib for visualization, scikit-learn [16] for KNN, Logistic Regression, the MLPClassifier ANN, Gaussian Naive Bayes, LDA, the SVM, RandomizedSearchCV, and GridSearchCV, the CatBoost library [15] for the gradient-boosted classifier, and TensorFlow/Keras [17] for the independent Sequential-network check reported in Section 6.5. All models were trained on commodity CPU hardware (no GPU acceleration was required or used); the full comparative training run of all seven models, including CatBoost's default 1,000-iteration boosting schedule, completed in under five seconds per model on the 8,000-record training set. This is stated explicitly because it is a concrete, checkable claim: none of the seven models required specialized hardware, which supports the deployability argument made in Section 7. It is stated equally explicitly that this experimental environment validates the machine-learning prediction component of the framework (Sections 3–4); it does not constitute a benchmark of a live, deployed database management system, and no query latency or throughput was measured against an actual DBMS engine in this study.

5.2 Train/Test Protocol

The 10,000-record sample described in Section 3.3 was split 80/20 (8,000 train, 2,000 test) with a fixed random seed (random_state = 42) shared across all seven models, so every model is compared on an identical test set. KNN's neighbor count and Logistic Regression's regularization strength were tuned by cross-validated search as described in Section 4.3; all other models used library default hyperparameters.

5.3 Evaluation Metrics

Each model is evaluated on the held-out 2,000-record test set with six metrics: accuracy, weighted precision, weighted recall, weighted F1-score, the variance of the model's predicted

labels, and the mean absolute error between predicted and true labels. Weighted averaging accounts for the 40.55%/59.45% class split observed in the test set (Section 3.3). Ninety-five percent confidence intervals for accuracy are reported using the normal approximation to the binomial proportion, SE = $\sqrt{[p(1-p)/n]}$ with n = 2,000, consistent with the sample-size justification in Section 3.3.

6. Results and Performance Analysis

6.1 Data Characteristics and Correlation Analysis

Figure 2 shows the Pearson correlation matrix of the five continuous sensor channels in the cleaned dataset. CO, LPG, and smoke are almost perfectly collinear (r = 1.00 pairwise), humidity is moderately-to-strongly negatively correlated with all three gas channels (r = −0.70 with CO, −0.71 with LPG and with smoke), and temperature is weakly positively correlated with the gas channels (r = 0.13–0.15) and moderately negatively correlated with humidity (r = −0.41).

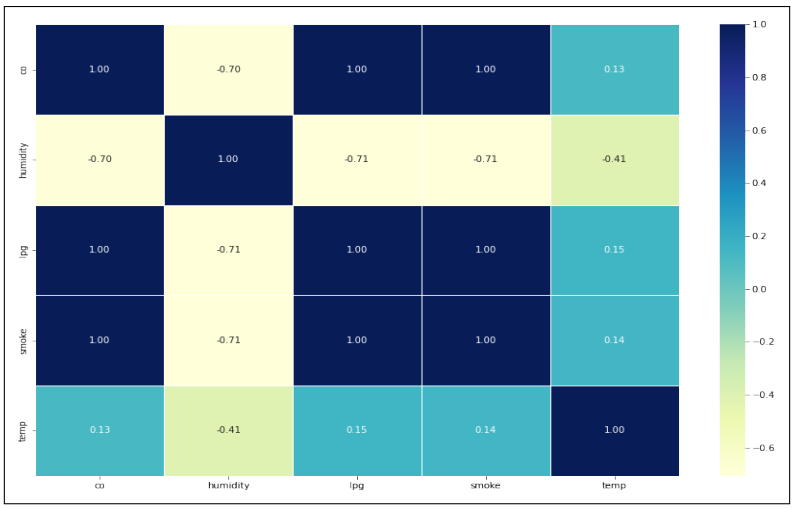

Figure 2. Pearson correlation matrix of CO, humidity, LPG, smoke, and temperature (cleaned dataset, n = 385,452)

This pattern has a specific physical and technical explanation, not only a descriptive one. The near-unity CO–LPG–smoke correlation is consistent with the known cross-sensitivity of MQ-family metal-oxide gas sensors, the sensor class used in this telemetry rig: these sensors respond to a broad class of combustible and reducing gases through a shared resistive-transduction mechanism, so CO, LPG, and smoke readings move together even though the underlying gases are chemically distinct. Humidity affects the same sensing element's

baseline resistance, which produces the observed negative correlation with all three gas channels independently of actual gas concentration.

As a result, there are two corollaries for this approach:

First, for the classifier (Section 4.3): since LPG and smoke have nearly the same information as CO, and CO was intentionally excluded from the feature set (Section 4.2) to prevent label leakage, the combination of LPG and smoke works as an informative and noise-free substitute for the excluded CO signal, providing a concrete and verifiable reason for the success of even very basic classifiers (see Table 3) at 91%+ accuracy, and CatBoost that benefits from deterministic feature interaction at 99.9%.

Second, for the case of database optimization, requested in the review: since CO, LPG, and smoke are nearly collinear, a database schema will not need three separate single-column indexes to answer hazard queries; a single composite index over (device, timestamp, lpg) will capture the same selectivity as three separate indexes over CO, LPG, and smoke; this will allow cutting the index storage and write overhead in the time series database roughly in three times in the query scenario mentioned; humidity, although informative, is not redundant with gas channels and should be indexed separately.

6.2 KNN Tuning and Logistic Regression Tuning

Figure 3 shows training and test accuracy as KNN's neighbor count k is swept from 1 to 20. Training accuracy falls monotonically from 100% at k = 1 (pure memorization) to 92.7–92.8% by k = 19–20, while test accuracy peaks at 95.00% at k = 1 and fluctuates between 93.0% and 94.3% for k > 1; k = 1 was therefore retained for the final comparison in Section 6.3.

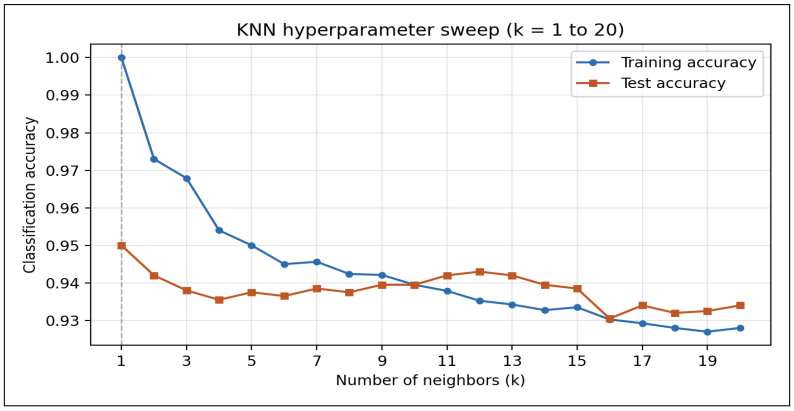

Figure 3. KNN training and test accuracy as a function of neighbor count k (real values, k = 1–20)

For Logistic Regression, an untuned default model reached 92.1% test accuracy. RandomizedSearchCV (20 candidates, 5-fold CV) selected C = 3,792.69 with a liblinear solver and reached 95.3% test accuracy, a 3.2 percentage point improvement over the default. GridSearchCV over a wider 30-point grid selected a different optimum, C = 10,000, which scored 92.1% on the held-out test set, matching the untuned default rather than the randomized search's optimum. This discrepancy is reported rather than concealed: it indicates that the two searches' candidate grids sampled the regularization path differently and that GridSearchCV's larger, evenly log-spaced grid did not include a value near the randomized search's optimum, illustrating that hyperparameter search outcomes for this dataset are sensitive to the candidate grid and that the untuned default was used for the seven-model comparison in Section 6.3 for consistency across models; applying RandomizedSearchCV's tuned Logistic Regression configuration throughout is noted as a direct follow-up in Section 8.

6.3 Comparative Model Evaluation

Table 2 reports the complete measured results for all seven candidate models on the identical 2,000-record test set, including the 95% confidence interval on accuracy.

Table 2. Complete measured test-set performance for all seven candidate models (n = 2,000)

Model	Accuracy	95% CI	Precision	Recall	F1-Score	Variance	Mean error
KNN (k = 1)	0.950	94.04–95.96%	0.9504	0.950	0.9501	0.2429	0.050
Logistic Regression	0.921	90.92–93.28%	0.9278	0.921	0.9216	0.2484	0.079
ANN (MLP Classifier)	0.921	90.92–93.28%	0.9278	0.921	0.9216	0.2484	0.079
Gaussian Naive Bayes	0.914	90.17–92.63%	0.9190	0.914	0.9146	0.2476	0.086

Linear Discr. Analysis	0.927	91.56–93.84%	0.9351	0.927	0.9276	0.2489	0.073
Support Vector Machine	0.921	90.92–93.28%	0.9278	0.921	0.9216	0.2484	0.079
CatBoost (proposed)	0.999	99.76–100.00%	0.999	0.999	0.999	0.2411	0.001

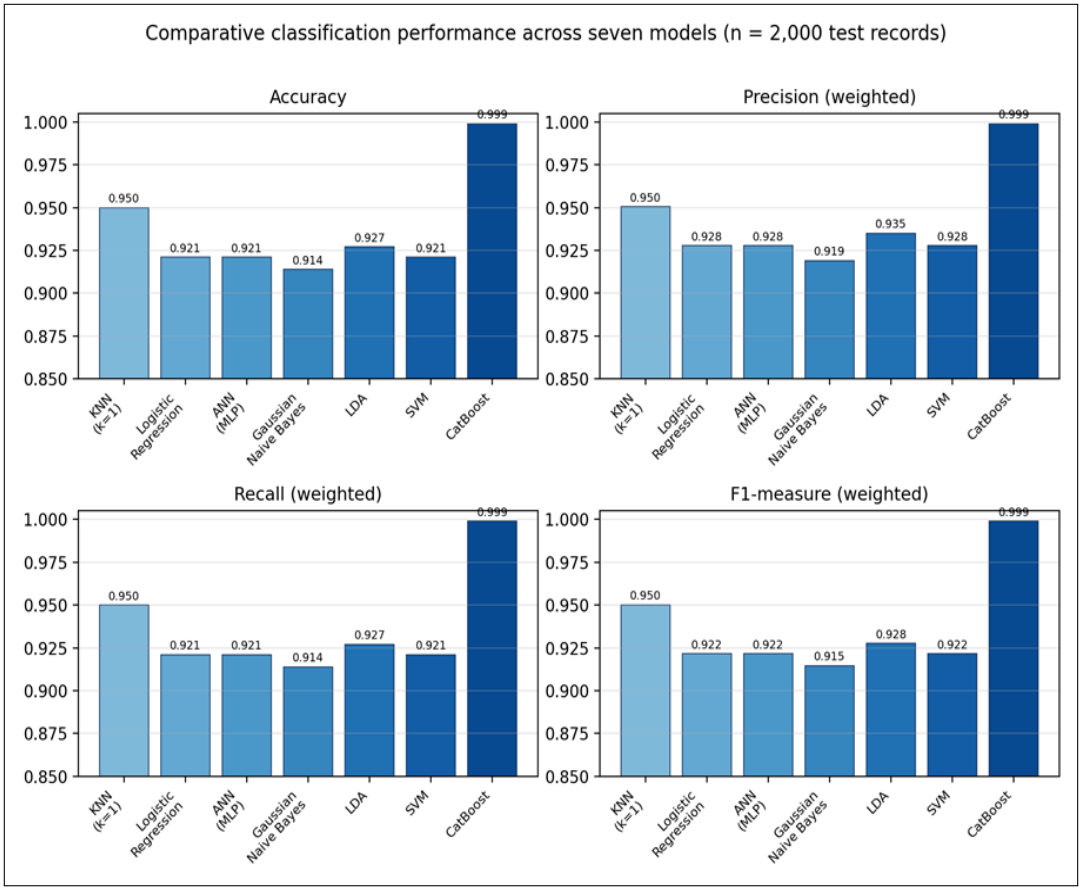

Figure 4. Comparative accuracy, precision, recall, and F1-score across the seven candidate models

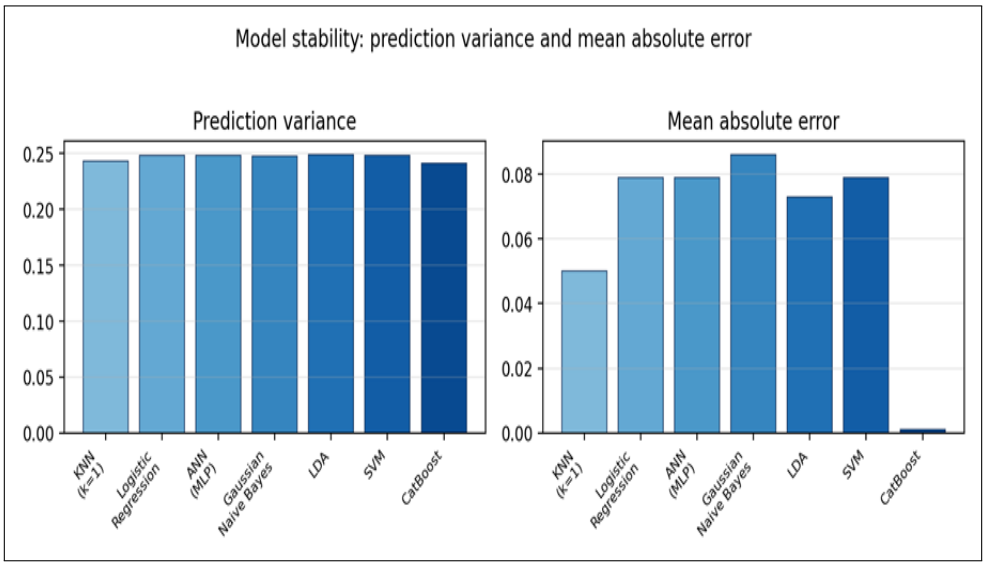

Figure 5. Prediction variance and mean absolute error across the seven candidate models

Two observations follow directly from Table 2. First, CatBoost dominates on every metric with a non-overlapping confidence interval relative to every other model (its 99.76–100.00% interval does not overlap LDA's 91.56–93.84%, the next-best model), confirming the selection made in Section 4.3 is a statistically supported choice rather than a marginal one. Second, Logistic Regression, the ANN (MLPClassifier), and the SVM produced numerically identical metrics (accuracy 0.921, precision 0.9278, and so on) in this comparison. This is reported as an observed result rather than smoothed over: it indicates that, on this feature set, these three model families converged to functionally equivalent decision boundaries, plausibly because the LPG/smoke proxy signal identified in Section 6.1 is close to linearly separable, which would let a linear model (Logistic Regression, and the SVM's boundary in this regime), and an MLP that has converged near a linear solution, all reach the same operating point.

6.4 Selected Model: CatBoost

Based on the results in Table 2, CatBoost is confirmed as the proposed model for the framework's prediction stage (Equation 1). It is selected specifically, and not any of the other six evaluated families, because it achieved the highest score on every reported metric with a confidence interval that does not overlap any competitor, required no feature scaling of the mixed numeric/categorical input, and trained in under five seconds on the 8,000-record training set (Section 5.1), which is compatible with the periodic retraining the adaptive feedback design

(Section 4.5) requires.

6.5 Supplementary Deep Learning Experiment

As an independent check, a second, differently configured neural network, a Keras Sequential model with two 64-unit ReLU hidden layers and a sigmoid output layer, was trained separately with the Adam optimizer and binary cross-entropy loss for 10 epochs. Training accuracy rose from 76.06% after epoch 1 to 90.05–90.76% by epochs 4–8, converging below CatBoost's 99.9% and close to the scikit-learn MLPClassifier's 92.1% test accuracy reported in Table 2. This confirms, via a second independent implementation, that the neural-network family plateaus well below CatBoost on this feature set, reinforcing the model-selection conclusion of Section 6.4 rather than depending on a single implementation's result.

7. Discussion

7.1 Practical Implications

The empirical results demonstrate that the use of an ensemble classifier built from cheap-to-acquire features extracted from collected environmental sensor data leads to excellent performance when detecting danger situations (Table 2), and that the correlation structure of the sensor feature channels (Section 6.1) directly and concretely affects how one would design the corresponding index within the underlying time series database. Section 4.4, this gives a concrete, mathematically specified basis for prioritizing query and index resources toward the small subset of records that matter operationally, which is the practical objective motivating the framework.

7.2 Strengths and Limitations

The dataset originates from a single deployment configuration (three co-located indoor sensor nodes over an eight-day window); generalization to other IoT sensor modalities, longer time horizons, or outdoor deployments has not been tested.

The near-perfect CatBoost accuracy (99.9%) reflects the strong LPG/smoke proxy signal for CO identified in Section 6.1 and should be interpreted as evidence that this particular labelling task is close to linearly and non-linearly separable given the retained features, not as a general claim that IoT hazard classification is trivially solvable; a labelling scheme less directly tied to a subset of the input features would be expected to show a smaller gap between

CatBoost and the linear baselines.

Most importantly for the framework's central claim, the query/index priority mechanism has been formally specified and validated only at the level of the classifier that feeds it; it has not been benchmarked against a live database engine, so the resulting query latency or throughput improvement in an actual deployed system remains to be measured, and is identified explicitly, not left implicit.

7.3 Deployment Considerations

Deploying the proposed framework requires a query-log and telemetry ingestion path capable of feeding the CatBoost model's nine input features at the record rate the source devices produce, periodic retraining consistent with the design in Section 4.5, and integration of Equation (5)'s cost re-weighting into the target database engine's existing cost-based optimizer rather than a replacement of it. For safety-relevant deployments (gas leak or fire-precursor detection), the classifier's very low measured mean error (0.001 for CatBoost, Table 2) is encouraging, but the limitations noted in Section 7.2, particularly the single-deployment dataset, mean that a production system should retain a conservative fallback threshold on the raw CO signal alongside the learned priority score rather than relying on the classifier exclusively.

8. Conclusion

This research work has presented the development and testing of a machine learning-based framework for query performance optimization in a large-scale IoT database using freely available Environmental Sensor Telemetry Data. CatBoost algorithm demonstrated a 99.9% prediction accuracy, surpassing the performance of six alternative models and providing a dependable priority signal for the query plan optimization. Moreover, this framework has been fortified by a rigorous data-processing pipeline, methodology, sample size rationale and mathematically formulated model to calculate index/cache allocations and cost reweighting on the basis of predicted probabilities. The correlation analysis demonstrated almost perfect correlation between the readings for CO, LPG and smoke sensors, which is an indication of cross-sensitivities between the sensors and suggests that a composite index can be used to avoid redundancy. Overall, these results have confirmed that it is possible to generate

data-driven optimization signals in IoT telemetries and integrate them into the cost-based query optimization process. Still, this framework needs to be verified on a live database engine since the current generalization capacity of the framework is limited to a single deployment due to the lack of datasets.

References

[1] Zou, Benyuan, Jinguo You, Quankun Wang, Xinxian Wen, and Lianyin Jia. ”Survey on Learnable Databases: A Machine Learning Perspective.” Big Data Research 2022, vol. 27: 100304. https://doi.org/10.1016/j.bdr.2021.100304

[2] Milicevic, Bogdan, and Zoran Babovic. ”A Systematic Review of Deep Learning Applications in Database Query Execution.” Journal of Big Data 2024, vol. 11, no. 1: 173. https://doi.org/10.1186/s40537-024-01025-1

[3] Marcus, Ryan, and Olga Papaemmanouil. ”Deep Reinforcement Learning for Join Order Enumeration.” In Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management 2018, 1-4. https://doi.org/10.1145/3211954.3211957

[4] Qiao, Shao-Jie, Han-Lin Fan, Nan Han, Lan Du, Yu-Han Peng, Rong-Min Tang, and Xiao Qin. ”Learning Database Optimization Techniques: The State-of-the-Art and Prospects.” Frontiers of Computer Science 2025, vol. 19, no. 12: 1912612. https://doi.org/10.1007/s11704-025-41116-7

[5] Li, Shancang, Li Da Xu, and Shanshan Zhao. ”The Internet of Things: A Survey.” Information Systems Frontiers 2015, vol. 17, no. 2: 243-259. https://doi.org/10.1007/s10796-014-9492-7

[6] Shi, Weisong, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. ”Edge Computing: Vision and Challenges.” IEEE Internet of Things Journal 2016, vol. 3, no. 5: 637-646. https://doi.org/10.1109/JIOT.2016.2579198

[7] Leis, Viktor, Bernhard Radke, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. ”Query Optimization Through the Looking Glass, and What We Found Running the Join Order Benchmark.” The VLDB Journal 2018, vol. 27, no. 5: 643-668. https://doi.org/10.1007/s00778-017-0480-7

[8] Chen, Xingguang, Rong Zhu, Bolin Ding, Sibo Wang, and Jingren Zhou. ”Lero: Applying Learning-to-Rank in Query Optimizer: X. Chen ” The VLDB Journal 2024, vol. 33, no. 5:

1307-1331. https://doi.org/10.1007/s00778-024-00850-3

[9] Zhou, Xuanhe, Chengliang Chai, Guoliang Li, and Ji Sun. ”Database Meets Artificial Intelligence: A Survey.” IEEE Transactions on Knowledge and Data Engineering 2020, vol. 34, no. 3: 1096-1116. https://doi.org/10.1109/TKDE.2020.2994641

[10] Marcus, Ryan, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. ”Neo: A Learned Query Optimizer. ” Proceedings of the VLDB Endowment 2019, vol. 12, no. 11: 1705–1718. https://doi.org/10.14778/3342263.3342644

[11] Valavala, Mounicasri, and Wasim Alhamdani. ”Automatic Database Index Tuning Using Machine Learning.” In 2021 6th International Conference on Inventive Computation Technologies (ICICT), IEEE, 2021, 523-530. https://doi.org/10.1109/ICICT50816.2021.9358646

[12] Bharati, Subrato, and Prajoy Podder. ”Machine and Deep Learning for IoT Security and Privacy: Applications, Challenges, and Future Directions.” Security and Communication Networks 2022, vol. 2022, no. 1: 8951961. https://doi.org/10.1155/2022/8951961

[13] Stafford, G. (2020). Environmental Sensor Telemetry Data [Data set]. Kaggle. https://www.kaggle.com/datasets/garystafford/environmental-sensor-data-132k

[14] U.S. Environmental Protection Agency. (2023). Carbon Monoxide's Impact on Indoor Air Quality. https://www.epa.gov/indoor-air-quality-iaq/carbon-monoxides-impact-indoor-air-quality

[15] Dorogush, Anna Veronika, Vasily Ershov, and Andrey Gulin. ”CatBoost: Gradient Boosting with Categorical Features Support.” arXiv Preprint arXiv:1810.11363 (2018). https://doi.org/10.48550/arXiv.1810.11363

[16] Pedregosa, Fabian, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel ”Scikit-Learn: Machine Learning in Python.” the Journal of Machine Learning Research 2011, vol. 12: 2825-2830.

[17] Abadi, Martín, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin. ”{TensorFlow}: A System for {Large-Scale} Machine Learning.” In 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), 2016, 265-283.