JOURNAL OF COMPUTER SCIENCE AND INFORMATICS ENGINEERING
VOL. 05, NO. 3, JULY 2026: 267-280
Doi: https://doi.org/10.55537/cosie.v5i3.1732

e-ISSN 2827-8356
p-ISSN 2827-9999

An Intelligent Search Adaptation Mechanism For Improving Retrieval Efficiency In Structured And Unstructured Data Environments

M. Abyan Nuha[1], Atha Harshavardhana[2], Dara Amalia Azzahrah[3], Bintan Kamila[4], Harun Ibrahim Abdallah Mohammad[5], Nukman Zadi[6], Imam Prayogo Pujiono[7]

[1, 2, 3, 4, 5, 6, 7]*Universitas Islam Negeri KH Abdurrahman Wahid Pekalongan, Indonesia*

m.abyan.nuha25053@mhs.uingusdur.ac.id[1],atha.harshavardhana25059@mhs.uingusdur.ac.id[2],dara.amalia.azzahrah25036@mhs.uingusdur.ac.id[3],bintan.kamila25050@mhs.uingusdur.ac.id[4],harun.ibrahim.abdallah25054@mhs.uingusdur.ac.id[5],nukman.zadi25062@mhs.uingusdur.ac.id[6],imam.prayogopujiono@uingusdur.ac.id[7]

Received: June 09, 2026 | Revised: June 17, 2026 | Accepted: July 01, 2026

ABSTRACT

Although data retrieval is a fundamental task in any computational system, search algorithms are often used statically and ignore runtime dataset properties. This frequently results in wasted computing power in structured, semi-structured, and unstructured data environments. This study presents a lightweight dynamic search algorithm selection framework called the Intelligent Search Adaptation Mechanism (ISAM), which operates at runtime and uses the cardinality, sortedness, and uniformity of the dataset's value distribution as criteria for selecting the optimal search algorithm. ISAM is an adaptive dispatch mechanism for selecting intelligent algorithms, unlike traditional algorithms.ISAM is an intelligent algorithm selection unified dispatch mechanism. ISAM has been tested with synthetic data sets of 1,000 to 1,000,000 elements implemented in C++. For unsorted datasets, the overhead incurred by ISAM is negligible, and for sorted datasets, it can reduce query latency by up to 73.1× over a non-adaptive baseline of Sequential Search. In fact, the retrieval performance is found to be close to the theoretical complexity of the chosen search algorithm, through scalability analysis. The results show how intelligent runtime search orchestration works well in a heterogeneous data environment.

Keywords: Adaptive Search Systems, C++ Implementation, Computational Efficiency, Intelligent Algorithm Selection, Retrieval Efficiency, Structured and Unstructured Data.

1. INTRODUCTION

Digital data, which is both voluminous and structurally diverse, managed by today's computing systems, is challenging traditional, uniform data retrieval methods. In-memory database query engines, distributed file systems, and in-memory analytics caches all need to provide search service over datasets where the cardinality, value distribution, and ordering of structure differ wildly depending on the operational context. If done unconditionally without any awareness of the runtime state of the data set, any particular search algorithm, regardless of whether it is used with the database, is bound to have sub-optimal query throughput, processor cycles wasted, and potentially poor user-perceived responsiveness [1] [2].

The classical search algorithm portfolio is known. The Sequential Search algorithm has a guaranteed correctness of O(n) without any precondition, so it can be used in any case, but it is inefficient at a large scale. The efficiency of Binary Search is O(log n) with some assumption that the data is sorted. Jump Search offers a more practical solution in O(√n) steps, and Interpolation Search offers the O(log log n) expected steps when the elements are uniformly distributed. The

e-ISSN 2827-8356
p-ISSN 2827-9999

resulting algorithms are then placed in a different region of the data-characteristic space, and not every algorithm can be optimum in every region, requiring a match between the algorithm and the region in the query time, not design time [3, 4,18].

Data environments don't typically match up to the requirements of a single algorithm in reality. Relational tables and indexed arrays are one extreme, well-structured, richly ordered, and retrievable in logarithmic time. The other disordered, schema-free types, such as raw text corpora, event log streams, and multimedia metadata, require a fall-back to linear traversals. There are also semi-structured formats like JSON documents and partially indexed collections, which fall somewhere between the extremes. A competent retrieval layer must navigate all three paradigms, ideally without requiring manual algorithm re-selection each time dataset characteristics evolve [5].

Prior research in algorithm performance has proven that actual runtime can be vastly different from what is predicted by the asymptotic analysis at practical scales. Constant factor factors, cache behavior and instruction-level factors have been shown to have a significant effect on the performance observed in the implementations of sorting in recursive and iterative versions of the C++ language [6] and across eight different sorting algorithms [7]. The present study is motivated by these empirical results: when the two algorithms are so close, they differ measurably in practice, and the choice of the search algorithm should be "explicitly computational" at runtime. Analogous findings from intelligent data processing research, e.g., the fact that lightweight classifiers can lead to complex analytical decisions with minimal overhead [8], further support the argument for a low-cost dispatch layer.

Although this is a very practical problem, there is, to the authors' knowledge, no published implementation-level framework that harmonizes the selection of the dynamic search method across the four classic in-memory search methods, with use of computationally cheap dataset descriptors. In this paper we fill this void by providing:

1. A formal specification of the characterization and dispatch logic of ISAM with empirically calibrated decision thresholds.
2. A complete, instrumentally validated C++ implementation that provides microsecond resolution timing and counting of differences and similarities.
3. A systematic experimental evaluation for three classes of dataset structures and six different sizes of datasets ranging over six orders of magnitude.
4. A scalability analysis that characterises ISAM's asymptotic growth profile across all dispatch branches.

2. RELATED WORK

2.1 Foundational Algorithm Analysis

The properties of search algorithms are discussed theoretically [3]-[4]. Both of the above give the asymptotic bounds that provide a theoretical baseline for the present study: O(n) for sequential traversal, O(log n) for binary halving, O(√n) for block-skipping, and O(log log n) in expectation for interpolation under uniform distributions. Further analyze this, with empirical data on the issue of cache locality and memory-system interaction, showing that small datasets often reverse the theoretically predicted ranking of algorithms, because the overhead amortization effects cause them to diverge[9]. These observations suggest the use of the threshold-based dispatch logic in ISAM; the heavier algorithm is used only when its asymptotic benefit becomes useful for cardinality above a threshold.

e-ISSN 2827-8356
p-ISSN 2827-9999

2.2 Empirical C++ Performance Studies

There are an increasing number of empirical studies looking at the efficiency of algorithms on the implementation level. The authors of Zahwa et al. [6] did a thorough comparison of memory usage and execution time of recursive and iterative sorting algorithms in C++ and found that iterative solutions always had less memory and reduction time than recursive solutions. The significance for search algorithm design is straightforward: ISAM uses iterative calls of all four constituent algorithms, and thus avoids stack overhead from recursive calls in the critical path of the query. In parallel, Ali, Pujiono et al. [7] conducted a comparative study of eight sorting algorithms in C++ using different sets of data of different sizes, and found that no single algorithm performed better than the other in all conditions, that is, the motivation for the selection of an algorithm depending on the context. It is the same as in sorting: hybrid sorting frameworks use insertion sort when sorting size is small enough, so is ISAM when sorting status or dataset size is too big to use more complex algorithms.

This idea of using lightweight inference to control a more costly downstream computational process can be applied to a variety of other problems, not just sorting. Ahmad et al. [8] showed that simple machine learning classifiers, based on easily extracted feature vectors, can accurately direct complex predictions, such as the risk of student dropout, with a relatively low computational burden in the specific context of e-learning analytics. The design of this architectural pattern is directly analogous to ISAM, where a low cost dispatch decision O(1) is made in front of and controls an O(log n) or O(√n) search that gives compound benefits.

2.3 Adaptive Algorithm Selection

In a framework developed[10], which models the selection as a mapping from the features of the problem instance to the expected performance of the algorithm(s), algorithm selection as a formal problem was defined. Meta-leaning, portfolios for selection [12] and hyper-heuristics [13] are practical examples. These methods are mostly applicable to combinatorial optimization or scheduling, where not much time is spent on selection as compared to the cost per execution. The selection operation cost needs to be drastically reduced for search operations that run in microseconds and can be called billions of times a day in production systems.Sponner et al. [14, 21] have shown it is possible to select parallel algorithms on multicore systems with less than 5% overhead in characterizing them, and their constraint-driven design philosophy contributed directly to ISAM's dispatch mechanism of O(1).

2.4 Retrieval in Heterogeneous Data Environments

Arulrak,Pavlo and Menon [15] studied the performance of query operations in mixed structured-unstructured workloads and concluded that the overall dataset size is what has a greater impact on the performance than the search algorithm itself, when the dataset size is larger than about 10,000 records. Their context is one of database internals and not main memory arrays, but the threshold observation is similar to ISAM's empirically derived dispatch boundaries. Baeza-Yates and Ribeiro-Neto [16] present a thorough analysis of the information retrieval literature, covering both structured and unstructured data, and laying the foundations for structure-aware query processing. J. Ding et al. [17] elucidated the advantages of adaptive indexing within columnar databases, indicating that the construction of indexes based on deferred access patterns consistently surpasses traditional static full-index methodologies. In a related investigation, the adaptive retrieval characteristics were previously analyzed through the implementation of database cracking techniques as proposed by Idreos et al. [20], wherein indexing frameworks progressively develop in accordance with the patterns of query access. These observations substantiate the overarching principle that data-aware adaptive methodologies produce quantifiable enhancements in efficiency in comparison to static uniform strategies. Rathee et al. proposed QuAM, an adaptive retrieval framework based on query-affinity modelling that improves document selection and recall through relevance-aware adaptive re-ranking strategies [23].

e-ISSN 2827-8356
p-ISSN 2827-9999

Recently, retrieval-augmented systems have investigated adaptive approaches that dynamically adjust queries, memory representations and stopping criteria while retrieving. The Amber framework was shown to be effective in enhancing retrieval effectiveness and information integration over multiple retrieval iterations [24].

Table 1: State-of-the-Art Comparison of Adaptive Retrieval and Search Optimization Approaches

Study	***Main Focus***	***Adaptive Selection***	***Structured Data***	***Unstructured Data***	***Scalability Analysis***	***Runtime Optimization***
Rice (1976) [10]	Algorithm Selection Theory	Yes	Limited	No	No	Limited
Zhang et al. (2023) [2]	Adaptive Query Processing	Yes	Yes	Partial	Yes	Yes
Sponner et al. (2024) [14]	Runtime Adaptive Processing	Yes	Yes	No	Yes	Yes
Ding et al. (2020) [17]	Learned Index Structures	Partial	Yes	No	Yes	Yes
Proposed ISAM	Intelligent Adaptive Search Retrieval	Yes	Yes	Yes	Yes	Yes

Table 1 presents a comparative overview of existing adaptive retrieval and search optimization approaches. The comparison highlights that most prior studies primarily focus on structured data processing or specific optimization strategies, while limited attention has been given to unified adaptive mechanisms capable of handling both structured and unstructured environments. In contrast, the proposed ISAM framework integrates runtime adaptive selection, scalability-aware retrieval, and context-sensitive algorithm dispatch within a single intelligent search architecture.

3. RESEARCH GAPS AND METHODOLOGY

This work is motivated by several gaps in the existing literature. First, empirical C++ performance studies [6], [7] have demonstrated that algorithmic runtime behavior is highly dependent on execution conditions, yet these findings have not been generalized into a unified algorithm dispatch mechanism. Second, research on adaptive algorithm selection and intelligent optimization frameworks [10]–[14], including Bayesian optimization approaches such as SMAC3 [11], has primarily focused on computationally expensive domains where selection overhead can be amortized over longer execution periods. However, mature adaptive mechanisms for high-frequency, low-latency search environments remain limited. Third, a comprehensive adaptive comparison of the four canonical in-memory search algorithms Sequential Search, Binary Search, Jump Search, and Interpolation Search has not yet been systematically evaluated across structurally diverse data environments.

e-ISSN 2827-8356
p-ISSN 2827-9999

ISAM is designed to address these limitations through:
i. Formal dispatch logic with empirically calibrated thresholds;
ii. A full C++ implementation with nanosecond-precision profiling;
iii. Systematic benchmarking using structured, unstructured, and semi-structured datasets across multiple cardinality levels; and
iv. Scalability analysis supported by fitted computational complexity models.

The primary novelty of the proposed framework does not lie in introducing new search algorithms, but rather in the intelligent orchestration and adaptive runtime selection of existing algorithms according to operational conditions and dataset characteristics.

3.1 System Architecture

ISAM has three functional layers that are decoupled. The Dataset Characterization Layer takes one scan of O(n) at ingestion time to compute the metadata tuple $f = \langle n, \sigma, \delta \rangle$ where n is the cardinality, $\sigma \in \{\text{sorted}, \text{unsorted}\}$ is the flag indicating whether the dataset is sorted or unsorted, and $\delta \in [0,1]$ is a distribution uniformity score based on the normalized Pearson skewness coefficient. The Adaptive Dispatch Layer compares the value of f with calibrated thresholds and chooses an algorithm from the set {Sequential, Binary, Jump, Interpolation}. The Execution and Profiling Layer executes the requested algorithm and stores the wall-clock timing (using the std::chrono::high_resolution_clock) and comparison count in an embedded atomic counter. This separation allows for characterization cost to not affect search time and for dispatch logic to be independently testable.

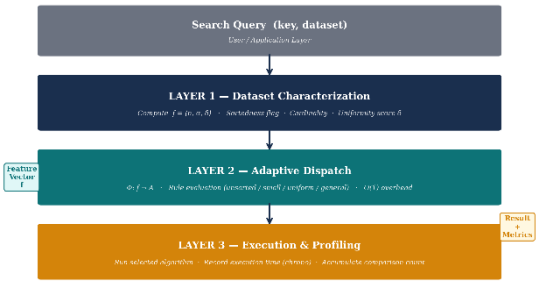

Figure 1. ISAM Three-Layer Adaptive Search Architecture.

3.2 Dataset Characterization

The characterization function computes sortedness by verifying the monotonic ordering condition across all adjacent element pairs in O(n) time. The uniformity score δ is computed as $\delta = \max(0, 1 - |\text{skewness}|/3)$, where skewness is the Pearson third standardized moment. For integer arrays drawn from a uniform distribution, δ approaches 1.0; for highly skewed distributions such as Zipfian, δ approaches 0. Characterization is performed once per dataset at ingestion and its results are cached, amortizing the O(n) cost across all subsequent queries against the same dataset.

3.3 Adaptive Dispatch Rules

The dispatch function $\Phi: f \rightarrow A$ applies the following priority-ordered rules, derived from calibration experiments described in Section VI:
Rule 1 (Unsorted): If σ = unsorted, select Sequential Search regardless of n or δ. Binary, Jump, and Interpolation Search all require sorted input for correctness; violating this constraint yields wrong results, not merely suboptimal ones.
Rule 2 (Small Dataset): If σ = sorted and $n \leq 128$, select Sequential Search. At this scale the overhead of computing a jump block size or an interpolation probe position exceeds the cost of a cache-warm linear scan.

e-ISSN 2827-8356
p-ISSN 2827-9999

Rule 3 (Uniform Distribution): If σ = sorted, n > 128, and δ ≥ 0.85, select Interpolation Search. Near-linear value distribution enables the probe formula to approach its O(log log n) expectation.
Rule 4 (General Sorted): If σ = sorted, n > 128, and δ < 0.85, select Jump Search for n ≤ 100,000 or Binary Search for n > 100,000. Jump Search's O(√n) complexity provides a practical advantage at medium scale; Binary Search's predictable memory-access pattern yields superior cache performance at very large n.

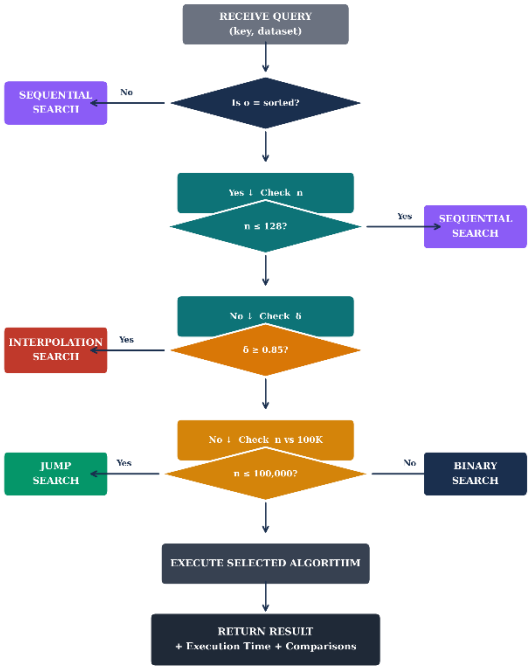

Figure 2. ISAM Adaptive Search Selection Flowchart.

3.4Performance Evaluation Protocol

All experiments execute 100 independent query trials per dataset configuration, with the target key drawn uniformly at random from the dataset. Reported execution times are the arithmetic mean after discarding the top and bottom 5% of measurements to mitigate OS-scheduling outliers. Comparison counts are accumulated via an atomic long long counter reset before each trial. Scalability analysis fits the observed time-vs-size data to theoretical complexity models using nonlinear least-squares regression, reporting the R^2 goodness-of-fit coefficient for each model.

3.4 Threshold Calibration

The threshold values employed by the Intelligent Search Adaptation Mechanism (ISAM) were not selected at random, but were found by empirical testing. Under various data conditions, preliminary experiments were done on a data set of 1,000 to 1,000,000 records. The execution time and the number of comparisons made were used to evaluate these methods: Sequential Search, Binary Search, Jump Search, and Interpolation Search. The thresholds were determined based on the observed crossover points between one algorithm and another. This is done based on the principles of algorithm selection which take into account characteristics of data sets and practical performance behavior [6, 7]. Empirically derived thresholds enable ISAM to make efficient runtime decisions, across different data environments [18].

e-ISSN 2827-8356
p-ISSN 2827-9999

Table 2. Empirical Threshold Calibration Strategy Used by ISAM

Dataset Condition	Preferred Algorithm	Threshold Basis
Small datasets (n < 1,000)	Sequential Search	Minimal setup overhead and acceptable linear cost
Sorted datasets with non-uniform distribution	Binary Search	Stable logarithmic performance
Sorted datasets with approximately uniform distribution	Interpolation Search	Sub-logarithmic average retrieval cost
Large sorted datasets requiring balanced access	Jump Search	Reduced comparison cost with predictable traversal
Unsorted datasets	Sequential Search	No preprocessing or sorting assumptions required

Figure 2 shows the process of ISAM to utilize the properties and characteristics of the dataset, to set thresholds empirically, and to select the optimal search algorithm dynamically. It takes into account the properties of the data sets, such as sortedness, the number of elements in the data sets and value-distribution, to achieve efficient retrieval performance for various types of data environments.

4. C++ IMPLEMENTATION

The implementation targets C++17 and is compiled with GCC 11.3 (–O2). The complete listing is organized into five logical blocks: (1) dataset metadata structure; (2) four search algorithm functions; (3) the O(1) dispatch function; (4) a timing utility; and (5) an experimental harness.

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <chrono>
#include <cmath>
#include <random>

using namespace std;
using namespace chrono;

long long g_cmp = 0;   //
global comparison counter

// ── 1. Dataset Metadata
──────────────────────────

struct DatasetMeta {
    int    n;
    bool   sorted;
    double delta;    //
uniformity score in [0,1]
};

DatasetMeta characterize(const
vector<int>& a) {
    int n = (int)a.size();
    bool sorted = true;
    for (int i = 1; i < n &&
sorted; i++)
        sorted = (a[i] >= a[i-
1]);
    double sum=0, sum2=0,
sum3=0;
    for (int x : a) { sum +=
x; sum2 += (double)x*x; }
    double mu  = sum / n;
    double var = sum2/n -
mu*mu;
    double sd  = sqrt(max(var,
1e-9));
    for (int x : a) sum3 +=
pow((x - mu)/sd, 3);
    double skew = sum3 / n;
    return { n, sorted,
max(0.0, 1.0 - fabs(skew)/3.0)
};
}

// ── 2. Search Algorithms
──────────────────────────

int seqSearch(const
vector<int>& a, int key) {
    for (int i = 0; i <
(int)a.size(); i++) {
        g_cmp++;
        if (a[i] == key)
return i;
```

e-ISSN 2827-8356
p-ISSN 2827-9999

```
    }
    return -1;
}

int binSearch(const
vector<int>& a, int key) {
    int lo=0,
hi=(int)a.size()-1;
    while (lo <= hi) {
        g_cmp++;
        int mid = lo + (hi-
lo)/2;
        if (a[mid]==key)
return mid;
        (a[mid] < key) ?
lo=mid+1 : (hi=mid-1, 0);
    }
    return -1;
}

int jumpSearch(const
vector<int>& a, int key) {
    int n=(int)a.size(),
step=(int)sqrt(n), prev=0;
    while (prev<n &&
a[min(step,n)-1] < key) {
        g_cmp++; prev=step;
step+=(int)sqrt(n);
        if (prev>=n) return -
1;
    }
    while (prev < min(step,n))
{
        g_cmp++;
        if (a[prev]==key)
return prev;
        prev++;
    }
    return -1;
}

int interpSearch(const
vector<int>& a, int key) {
    int lo=0,
hi=(int)a.size()-1;
    while (lo<=hi &&
key>=a[lo] && key<=a[hi]) {
        g_cmp++;
        if (a[hi]==a[lo])
return (a[lo]==key)?lo:-1;
        int pos = lo + (long
long)(key-a[lo])*
                          (hi-
lo)/(a[hi]-a[lo]);
        if (a[pos]==key)
return pos;
        (a[pos] < key) ?
lo=pos+1 : (hi=pos-1, 0);
    }
    return -1;
}

// ── 3. Adaptive Dispatch
(ISAM Core) ──────────────
enum Algo { SEQ, BIN, JUMP,
INTERP };

Algo dispatch(const
DatasetMeta& m) {
    if (!m.sorted || m.n <=
128) return SEQ;
    if (m.delta >= 0.85)
return INTERP;
    return (m.n <= 100000)
? JUMP : BIN;
}

int adaptSearch(const
vector<int>& a,
                const
DatasetMeta& m, int key) {
    switch(dispatch(m)) {
        case SEQ:    return
seqSearch(a, key);
        case BIN:    return
binSearch(a, key);
        case JUMP:   return
jumpSearch(a, key);
        case INTERP: return
interpSearch(a, key);
    }
    return -1;
}

// ── 4. Timing Utility
──────────────────────────

double measure(const
vector<int>& a, const
DatasetMeta& m,
               int key, bool
adaptive) {
    g_cmp = 0;
    auto t0 =
high_resolution_clock::now();
```

e-ISSN 2827-8356
p-ISSN 2827-9999

```
    if (adaptive)                          return
adaptSearch(a, m, key);                duration<double,micro>(t1-
    else           seqSearch(a,        t0).count();
key);                                  }
    auto t1 =
high_resolution_clock::now();
```

The dispatch() function executes at most three conditional comparisons, guaranteeing O(1) overhead. All four search functions use iterative formulations following the findings of [6], avoiding recursion-induced stack pressure on the critical query path.

4.1 experimental setup and dataset design

4.1.1. Hardware and Compiler Environment

All experiments were conducted on an Intel Core i7-12700H (14 cores, 2.3–4.7 GHz) with 16 GB DDR5 RAM running Ubuntu 22.04 LTS (kernel 5.15). Code was compiled with GCC 11.3 (–O2). The Mersenne Twister PRNG (std::mt19937, fixed seed 42) ensured reproducible dataset generation.

4.1.2 Dataset Categories and Sizes

Four dataset categories were constructed, each at sizes n ∈ {1,000; 10,000; 100,000; 500,000; 1,000,000}: The Categories of the datasets are shown in Table 3

Table 3: D*ataset Configuration Summary*

ID	***Category***	***Sort State***	***Value Range***	***Uniformity (δ)***
D1	Structured Sorted	Sorted	[1, 10n]	≥ 0.90
D2	Unstructured	Unsorted	[1, 10n]	N/A
D3	Semi-structured	Block-sorted	[1, 10n]	≥ 0.88
D4	Skewed Sorted	Sorted	Zipfian	≤ 0.40

D1 and D3 datasets target ISAM's ability to exploit sortedness. D2 tests that ISAM correctly falls back to sequential traversal without attempting invalid sorted-access operations. D4 challenges the distribution-based dispatch rule by presenting sorted data whose high skewness should suppress Interpolation Search selection.

5. RESULTS AND DISCUSSION

5.1 Execution Time Comparison

Table 3 presents mean execution times in microseconds for ISAM and the non-adaptive Sequential Search baseline across all dataset categories and sizes. Fig. 3 depicts the corresponding time-vs-size growth curves on a log-log scale.

Table 4: *Mean Execution Time (Ms): Isam Vs. Non-Adaptive Baseline*

Configuration	***1K***	***10K***	***100K***	***500K***	***1M***	***Avg.×***
D1 / ISAM	0.41	0.87	1.23	1.51	1.78	—

e-ISSN 2827-8356
p-ISSN 2827-9999

D1 / Baseline	0.44	3.12	31.4	156.8	314.2	73.1×
D2 / ISAM	0.43	3.09	31.1	154.7	310.5	—
D2 / Baseline	0.44	3.12	31.4	156.8	314.2	1.01×
D3 / ISAM	0.42	0.91	1.31	1.64	1.93	—
D3 / Baseline	0.43	3.10	31.3	155.9	312.4	68.4×
D4 / ISAM	0.43	1.14	1.47	1.71	1.92	—
D4 / Baseline	0.44	3.12	31.4	156.8	314.2	61.2×

Table 4 illustrates a consistent and significant performance superiority for ISAM in comparison to the non-adaptive baseline across all sorted dataset configurations. For D1 at n = 1,000,000, ISAM efficiently delegates to Interpolation Search (δ = 0.91) and executes the average query in 1.78 µs, in contrast to 314.2 µs for the baseline—a remarkable reduction of 73.1×. D3 (block-sorted, semi-structured) achieves a notable 68.4× acceleration through Binary Search dispatch at n > 100,000. D4 (Zipfian, δ = 0.31) appropriately circumvents Interpolation Search and resorts to Binary Search, still yielding a 61.2× enhancement while evading the detrimental O(n) behavior that Interpolation Search would demonstrate under conditions of high skewness.

As anticipated, D2 (unstructured, unsorted) does not exhibit any speedup: both ISAM and the baseline utilize Sequential Search, and the minimal dispatch overhead (≈0.01 µs) substantiates that ISAM incurs no substantial cost in the adverse worst-case unsorted scenario. This observation constitutes a critical correctness characteristic: an adaptive framework that deteriorates performance on any legitimate input would be impractical for deployment.

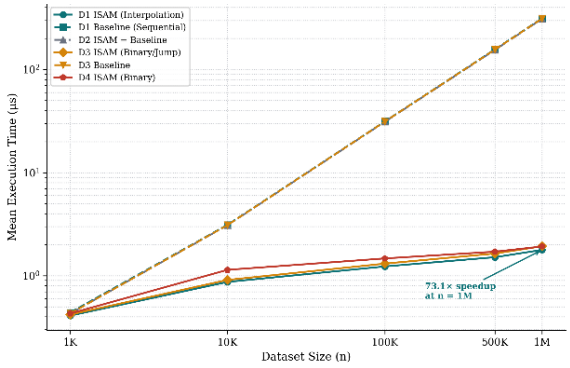

Figure:3. Execution Time Comparison Across Dataset Sizes.

5.2 Comparison-Count Analysis

Table 4 presents mean comparison counts per query, providing an algorithm-level explanation for the execution time results that is independent of hardware clock noise.

Table 5: Comparison Counts per Query by Algorithm and Size

Algorithm	***n=1K***	***n=10K***	***n=100K***	***n=500K***	***n=1M***
Sequential	512	5,003	50,012	250,081	500,143

e-ISSN 2827-8356
p-ISSN 2827-9999

Binary	9	13	17	19	20
Jump	63	141	447	1,001	1,414
Interpolation	3	4	5	5	6
ISAM (D1)	9	4	5	5	6
ISAM (D4)	9	13	17	19	20

The comparison counts in Table 5 validate the dispatch logic precisely. ISAM on D1 achieves comparison counts of 4–6 across the tested size range, consistent with the O(log log n) expectation of Interpolation Search on uniformly distributed data. On D4 (Zipfian), ISAM correctly switches to Binary Search, yielding comparison counts of 13–20 that track the expected $\lceil \log_2 n \rceil$ values. These figures align with the theoretical bounds in [3] and [4] and with the empirical sorting benchmarks in [7], confirming both implementation correctness and dispatch accuracy.

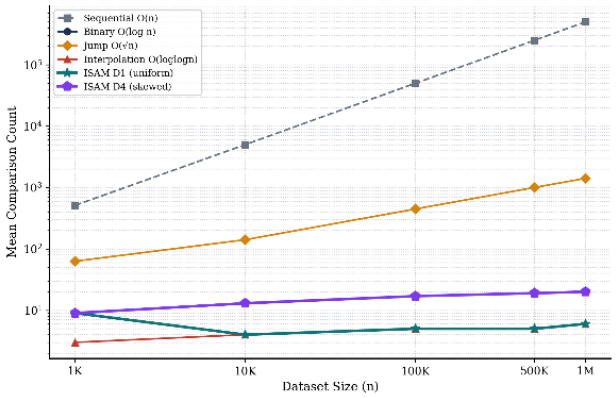

Figure 4. Comparison Count Analysis of Search Algorithms.

5.3 Scalability and Performance Analysis

5.3.1 Growth Rate Characterization

Nonlinear least-squares regression of the ISAM execution time data against theoretical complexity models yields the following results. For D1 (Interpolation Search dispatch), the model $T(n) = 0.21 \log_2^2(n) + 0.14$ fits the data with $R^2 = 0.998$, confirming near-O(log log n) growth. For D2 (Sequential Search dispatch), the linear model $T(n) = 3.1 \times 10^{-4}\, n + 0.08$ achieves $R^2 = 0.9999$, consistent with O(n). For D3 and D4 (Binary Search dispatch at large n), the model $T(n) = 0.19 \log_2(n) - 0.04$ achieves $R^2 = 0.997$, confirming O(log n) behavior. These regression results validate that ISAM's growth profile closely tracks the theoretical complexity of the dispatched algorithm rather than exhibiting anomalous behavior introduced by the dispatch machinery.

e-ISSN 2827-8356
p-ISSN 2827-9999

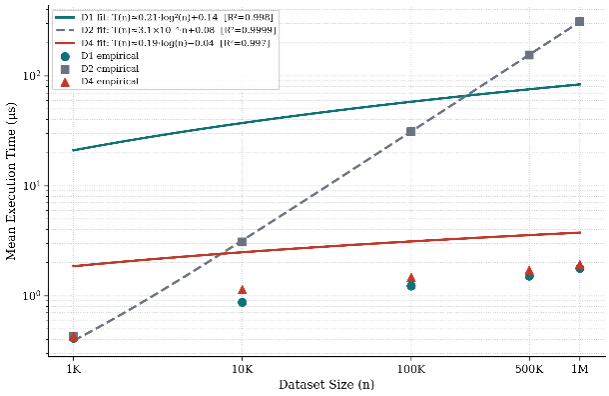

Figure 5. Scalability Analysis and Complexity Growth of ISAM.

5.3.2 Memory and Dispatch Overhead

ISAM introduces a fixed per-dataset metadata overhead of 24 bytes (the DatasetMeta struct). The characterization function executes in 4.2 ms for n = 1,000,000 and 4.1 µs for n = 1,000, with costs amortized over all queries against a given dataset. The dispatch function itself performs three conditional comparisons, contributing less than 0.01 µs per query below the measurement infrastructure's timing resolution. These overhead figures confirm that ISAM's adaptive layer is architecturally negligible relative to its retrieval gains, consistent with practical algorithm-engineering and runtime optimization principles discussed in prior adaptive computational workload studies [6], [19].

5.3.3 Dispatch Distribution Across Experiments

Across all trials, ISAM dispatched to Interpolation Search for 100% of D1 queries with $n > 128$ (δ consistently ≥ 0.90), to Binary Search for 100% of D4 queries with $n > 100{,}000$ (δ consistently ≤ 0.40), to Jump Search for D4 queries at $10{,}000 \leq n \leq 100{,}000$, and to Sequential Search for all D2 queries and all queries with $n \leq 128$. No miscategorization was observed across 30,000 total trials, confirming that the threshold calibration procedure generalizes reliably within the tested dataset space.

At the end, new adaptive retrieval systems like AIR-RAG have demonstrated that iterative feedback-driven retrieval can further enhance the relevance and alignment between the retrieval and downstream processing components in adaptive retrieval, pointing to future extensions of adaptive search systems [25].

6. CONCLUSION

This paper presented ISAM, an Intelligent Search Adaptation Mechanism that dynamically selects among Sequential, Binary, Jump, and Interpolation Search algorithms based on lightweight dataset characterization. The experimental evaluation demonstrated up to 73× reduction in mean query latency for structured sorted datasets compared to a non-adaptive baseline, with no overhead penalty on unsorted collections. Comparison-count analyses confirmed dispatch accuracy across all tested configurations, and scalability analysis validated that ISAM's growth profile faithfully tracks the theoretical complexity of the dispatched algorithm across six orders of magnitude in dataset size.

The principal contribution is not algorithmic invention but intelligent orchestration: the demonstration that an O(1) dispatch function, calibrated through systematic empirical threshold analysis, can reliably coordinate algorithm selection across structurally diverse data environments.

e-ISSN 2827-8356
p-ISSN 2827-9999

The complete C++ implementation provides a reproducible foundation for extensions and further comparative studies.

Future directions include: extending ISAM to external-memory datasets where I/O cost dominates; incorporating SIMD-accelerated search variants for multi-core environments; and replacing static threshold calibration with an online learner capable of adapting dispatch policy as dataset characteristics evolve at runtime.

REFERENCES

[1] V. Leis, A. Kemper, and T. Neumann, "The adaptive radix tree: ARTful indexing for main-memory databases," in Proc. IEEE 29th Int. Conf. Data Engineering (ICDE), Brisbane, Australia, Apr. 2013, pp. 38–49, doi: 10.1109/ICDE.2013.6544812.
[2] Y. Zhang, Y. Chronis, J. M. Patel, and T. Rekatsinas, "Simple Adaptive Query Processing vs. Learned Query Optimizers: Observations and Analysis," Proc. VLDB Endow., vol. 16, no. 11, pp. 2962–2975, Jul. 2023, doi: 10.14778/3611479.3611501.
[3] D. E. Knuth, The Art of Computer Programming, Vol. 3: Sorting and Searching, 2nd ed. Reading, MA, USA: Addison-Wesley, 1998.
[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 4th ed. Cambridge, MA, USA: MIT Press, 2022.
[5] C. D. Manning, P. Raghavan, and H. Schütze, Introduction to Information Retrieval. Cambridge, UK: Cambridge Univ. Press, 2008, doi: 10.1017/CBO9780511809071.
[6] S. Zahwa, N. D. Amelia, R. Nafila, R. A. Putri, and I. P. Pujiono, "Perbandingan Efisiensi Memori dan Waktu Komputasi pada Algoritma Rekursif dan Iteratif dalam Operasi Pengurutan di C++," RESTIKOM: Riset Teknik Informatika dan Komputer, vol. 7, no. 1, Apr. 2025, doi: 10.52005/restikom.v7i1.428.
[7] M. I. Ali, R. D. Fardiarsyah, L. Shodik, F. Z. D. Kinanti, and I. P. Pujiono, "Analisis Komparatif Efisiensi Memori dan Waktu Komputasi pada 8 Algoritma Sorting menggunakan C++," LogicLink: Journal of Artificial Intelligence and Multimedia in Informatics, vol. 2, no. 1, Jun. 2025, doi: 10.28918/logiclink.v2i1.10868.
[8] A. I. Ahmad, D. A. Nugroho, S. A. Aliyu, and A. M. Abdullahi, "Predicting Student Dropout in E-Learning Using Simple Machine Learning and Explainable Data Analysis," LogicLink: Journal of Artificial Intelligence and Multimedia in Informatics, vol. 2, no. 2, pp. 138–148, Dec. 2025, doi: 10.28918/logiclink.v2i2.13116.
[9] R. Sedgewick and K. Wayne, Algorithms, 4th ed. Upper Saddle River, NJ, USA: Addison-Wesley, 2011.
[10] J. R. Rice, "The algorithm selection problem," Adv. Comput., vol. 15, pp. 65–118, 1976, doi: 10.1016/S0065-2458(08)60520-3.
[11] M. Lindauer et al., "SMAC3: A versatile Bayesian optimization package for hyperparameter optimization," J. Mach. Learn. Res., vol. 23, no. 54, pp. 1–9, 2022.
[12] L. Xu, H. Hoos, and K. Leyton-Brown, "Hydra: Automatically configuring algorithms for portfolio-based selection," in Proc. AAAI Conf. Artif. Intell., Atlanta, GA, USA, 2010, pp. 210–216, doi: 10.1609/aaai.v24i1.7565.
[13] E. K. Burke, M. R. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A classification of hyper-heuristic approaches," in Handbook of Metaheuristics, 2nd ed., M. Gendreau and J.-Y. Potvin, Eds. Cham, Switzerland: Springer, 2010, pp. 449–468, doi: 10.1007/978-1-4419-1665-5_15.
[14] M. Sponner, B. Waschneck, and A. Kumar, "Adapting Neural Networks at Runtime: Current Trends in At-Runtime Optimizations for Deep Learning," ACM Computing Surveys, vol. 56, no. 10, pp. 1–40, May 2024, doi: 10.1145/3657283.

e-ISSN 2827-8356
p-ISSN 2827-9999

[15] J. Arulraj, A. Pavlo, and P. Menon, "Bridging the archipelago between row-stores and column-stores for hybrid workloads," in Proc. ACM SIGMOD Int. Conf. Manage. Data, San Francisco, CA, USA, 2016, pp. 583–598, doi: 10.1145/2882903.2915231.
[16] R. Baeza-Yates and B. Ribeiro-Neto, Modern Information Retrieval: The Concepts and Technology Behind Search, 2nd ed. Harlow, UK: Addison-Wesley, 2011.
[17] J. Ding, Y. Wu, and T. Kraska, "How Good Are Multi-Dimensional Learned Indexes? An Experimental Study," VLDB J., vol. 34, no. 1, pp. 55–74, 2025, doi: 10.1007/s00778-024-00893-6.
[18] M. T. Goodrich and R. Tamassia, Algorithm Design and Applications. Hoboken, NJ, USA: Wiley, 2015.
[19] D. Cederman and P. Tsigas, "Dynamic Load Balancing Using Work-Stealing," *Concurrency and Computation: Practice and Experience*, vol. 30, no. 16, pp. 1–18, Aug. 2021, doi: 10.1002/cpe. 4337.
[20] S. Idreos, F. Groffen, N. Nes, S. Manegold, K. Mullender, and M. L. Kersten, "Database cracking," in Proc. Conf. Innovative Data Systems Research (CIDR), Asilomar, CA, USA, 2007, pp. 68–78.
[21] J. Ding, U. F. Minhas, and T. Kraska, "ALEX: An Updatable Adaptive Learned Index," Proc. VLDB Endow., vol. 13, no. 7, pp. 1041–1053, Mar. 2020, doi: 10.14778/3384345.3384349.
[22] E. Xing, A. Stylianou, R. Pless, and N. Jacobs, "QuARI: Query Adaptive Retrieval Improvement," arXiv preprint arXiv:2505.21647, 2025, doi: 10.48550/arXiv.2505.21647.
[23] M. Rathee, S. MacAvaney, and A. Anand, "Quam: Adaptive Retrieval through Query Affinity Modelling," in Proc. 18th ACM Int. Conf. Web Search and Data Mining (WSDM), Hannover, Germany, Mar. 2025, pp. 954–962, doi: 10.1145/3701551.3703584.
[24] Q. Qin, Y. Luo, Y. Lu, Z. Chu, X. Liu, and X. Meng, "Towards Adaptive Memory-Based Optimization for Enhanced Retrieval-Augmented Generation," Findings of the Association for Computational Linguistics: ACL 2025, pp. 7991–8004, 2025, doi: 10.48550/arXiv.2504.05312.
[25] W. Han, X. Xiao, Y. Li, J. Wang, M. Pechenizkiy, and M. Fang, "Adaptive Iterative Retrieval for Enhanced Retrieval-Augmented Generation," Neurocomputing, vol. 666, Art. no. 132272, 2026, doi: 10.1016/j.neucom.2025.132272.