# A Fuzzy Logic-Driven Semantic and Binary Tree-Based Indexing Framework for Scalable IoT Data Storage and Retrieval

Khaled Halimi [1*], Abelhalim Hadjadj [2], Zineddine Kouahla[1], Brahim Farou[1]
[1]Department of Computer Science, LabSTIC Laboratory, University of Guelma 08 May 1945, Guelma, Algeria
[2]Abdelhafid Boussouf University Center, Mila, Algeria
E-mail: halimi.khaled@univ-guelma.dz, a.hadjadj@centre-univ-mila.dz, kouahla.zineddine@univ-guelma.dz, farou.brahim@univ-guelma.dz
*Corresponding Author

**Keywords:** internet of things, semantic web, fuzzy logic, indexing, data storage & retrieval, healthcare

**Received:** January 12, 2025

*The rapid growth of Internet of Things (IoT) devices presents significant data management challenges due to heterogeneity, interoperability issues, and massive data volumes, which hinder seamless data exchange and limit the IoT's potential. While the Semantic Internet of Things (SIoT) offers improvements through semantic web technologies, existing approaches often struggle with scalable data storage and efficient retrieval. To address this, the paper proposes a comprehensive, multi-layered architecture for efficient, scalable semantic IoT data handling. The architecture comprises: (1) an Edge Layer that utilizes the SAREF ontology to standardize heterogeneous device data into RDF format; (2) a Fog Layer performing fuzzy logic-based classification for enhanced data organization under uncertainty and binary tree-based indexing for efficient retrieval; and (3) a Cloud Layer for centralized storage. This approach integrates fuzzy logic for improved data categorization, particularly demonstrated through enhanced MEWS classification in healthcare, and a novel binary tree indexing method optimized for RDF file retrieval based on semantic content and fuzzy scores. Three dedicated algorithms govern the classification, indexing, and retrieval phases. Experimental validation using healthcare datasets demonstrates the framework's effectiveness. Specifically, the binary tree indexing reduces average retrieval times by orders of magnitude compared to non-indexed. Furthermore, the complete framework maintains stable and low query execution times (<0.01 s) even with 100,000 RDF files, significantly outperforming traditional RDF triple stores, which exhibit substantial performance degradation at scale. By significantly improving RDF data organization and retrieval efficiency, this work offers a scalable and innovative solution for managing Big IoT data, paving the way for advancements across various sectors.*

*Povzetek: Članek opisuje večplastno arhitekturo za SIoT, ki z uporabo semantike, mehke logike in binarnega indeksiranja omogoča učinkovito in razširljivo shranjevanje ter iskanje velikega RDF-podatkovja.*

## 1 Introduction

The growth of Internet of Things (IoT) devices is accelerating at an unprecedented rate. According to estimates from CISCO and Statista, by 2025, the number of connected devices will exceed 60 billion [1]. This surge in device connectivity has led to increasing challenges related to data heterogeneity and interoperability. Different sectors within the IoT ecosystem are producing their own devices, defining proprietary protocols, and developing isolated platforms. As a result, large amounts of heterogeneous data cannot be exchanged with different IoT platforms and applications. This lack of interoperability has led to the IoT domain losing more than 40% of its potential value [2].

To address these challenges, the Semantic Internet of Things (SIoT) has emerged as an effective approach by combining the Semantic Web with the Internet of Things. The SIoT framework is well-suited to managing the heterogeneity of data and enabling fluid communication between devices, making IoT data understandable and usable by machines. Using tools such as RDF, RDF Schema, OWL and Sparql, the SIoT standardizes the representation, processing and recovery of IoT data. This simplifies not only the exchange of information between different IoT-based systems, but also harnesses the full potential of IoT by transforming complex and heterogeneous data into meaningful and interconnected resources [3].

The Semantic Internet of Things is built on the basis of the Resource Description Framework (RDF) [4], which serves as its foundation for representing and generating output data. RDF is the standard format for semantically representing information on the Web, offering a structured and universally understandable framework. It relies on a static model composed of assertions and typically uses XML syntax to encode content. The essence of RDF lies in its use of triples—statements that describe relationships between resources. Each triple consists of a subject, predicate, and object, forming a simple yet powerful mechanism for connecting data meaningfully.

The SIoT primarily relies on RDF for data representation due to its flexible and adaptive structure.

Storing data in RDF format ensures that it remains preserved and accessible for future reference or use. However, the rapid increase in data generation has resulted in the exponential growth of RDF datasets, presenting significant challenges in storage and retrieval. For instance, the Linked Open Data (LOD) cloud now hosts 1,344 datasets with 16,308 interlinks [5]. Managing such vast amounts of RDF data poses critical issues, including efficient storage space allocation, real-time data processing, and the fragmentation of datasets. These challenges complicate retrieval processes and often result in irrelevant or inaccurate outcomes. While the SIoT holds significant promise, existing frameworks face some challenges, particularly in addressing the complexities of storing and retrieving RDF datasets [6]. To overcome these challenges, this paper proposes a comprehensive SIoT framework built on a multi-layered architecture grounded in core IoT principles. This architecture, designed to tackle the inherent ecosystem issues, consists of: an Edge Layer, designed to process data near the data sources. This layer incorporates a semantic model enhanced by the SAREF[1] ontology, resolving the issue of heterogeneity of data originating from various IoT devices by standardizing data generation and representation in RDF format. The framework also features a sophisticated fog layer that integrates advanced mechanisms such as fuzzy logic [7], clustering optimization [8], and binary tree [9] data indexing. These elements work together to improve data organization, retrieval efficiency, and overall system performance. This approach represents a significant step forward in addressing the storage and retrieval challenges of RDF datasets, paving the way for more efficient and scalable SIoT implementations.

The rest of the paper is organized as follows: Section 2 presents the research motivation. Section 3 reviews related work. Section 4 introduces the proposed framework. Section 5 presents a healthcare application scenario, while Section 6 describes the experimental setup and evaluation. Finally, Section 7 concludes the paper.

## 2 Research motivation

The rapid expansion of the Internet of Things (IoT) is transforming numerous sectors by enabling continuous connectivity among devices, sensors, and systems. However, the exponential growth in connected devices introduces pressing data management challenges—particularly in domains such as healthcare, smart cities, and industry—where vast, heterogeneous data streams are generated in real time. Traditional storage and retrieval systems often struggle to keep pace with this scale, leading to inefficiencies and poor interoperability across platforms. These challenges are rooted in both the diversity of data formats—ranging from simple numerical values to unstructured content—and the need for timely, accurate decision-making based on these data streams. Addressing these limitations requires robust frameworks capable of semantic integration, efficient classification, and scalable indexing.

This research is motivated by the need to develop a robust, scalable, and efficient framework to enhance IoT-based solutions by addressing their limitations in handling large-scale IoT data. Specifically, we investigate the incorporation of semantic technologies, fuzzy logic, and indexing techniques to improve the classification, storage, and retrieval of IoT data, thereby enabling more accurate and reliable decision-making in real-time applications.

To guide this investigation, we formulate the following primary research question (PRQ):

**PRQ:** To what extent does the proposed multi-layered SIoT architecture—integrating semantic representation (SAREF/RDF), fuzzy logic clustering, and binary tree indexing—improve the efficiency, scalability, and accuracy of managing and retrieving large-scale IoT data, compared to existing SIoT approaches and traditional RDF storage methods? This primary question leads to several specific inquiries:

- **RQ1:** How effectively can semantic technologies be leveraged within the Edge Layer to standardize heterogeneous IoT data representation and support scalable data management?
- **RQ2:** To what extent does the integration of fuzzy logic enhance the accuracy of classifying and organizing large volumes of RDF-represented IoT data within the Fog Layer, compared to methods based on crisp thresholds?
- **RQ3:** What level of performance improvement—measured in terms of speed and scalability—is achieved by using a binary tree-based indexing method for retrieving specific RDF files, compared to non-indexed search and conventional triple-store query mechanisms?
- **RQ4:** Does the proposed framework provide a viable and high-performing solution for managing real-world, large-scale IoT data, as demonstrated in a healthcare context?

Based on these research questions, we propose and aim to validate the following hypotheses:

- **H1:** The proposed multi-layered SIoT architecture will demonstrate significantly lower query execution times and improved scalability for RDF data retrieval, compared to baseline RDF management systems.
- **H2:** Incorporating fuzzy logic into the data classification process will lead to measurable improvements in classification accuracy and enable a more nuanced representation of data, compared to using standard crisp thresholds alone.
- **H3:** The binary tree-based indexing method will yield significantly faster data retrieval times for specific RDF files, compared to a linear search approach across the datasets.

By addressing these research questions and testing the associated hypotheses, this study aims to deliver a validated, efficient, scalable, and reliable solution for

[1] SAREF ontology https://saref.etsi.org/

managing the growing volume of IoT data—particularly in critical sectors such as healthcare—while supporting real-time decision-making and ensuring semantic consistency and accuracy.

## 3 Related works

The Internet of Things (IoT) has revolutionized how physical objects connect and interact, becoming indispensable across various domains. Connectivity between IoT devices is enabled by a network of sensors and actuators that work collaboratively to collect and exchange data seamlessly. To handle the vast amounts of data generated by these devices, various computational infrastructures are employed. Cloud computing provides the backbone for large-scale data storage and processing, while fog computing brings processing closer to the edge, reducing latency. In contrast, edge computing takes this a step further by enabling real-time data processing directly at the device level, thereby supporting faster and more efficient decision-making [10]. Despite its potential, IoT faces significant challenges—particularly in managing the large volume of data it generates and overcoming the complexity introduced by the diversity of devices and communication protocols. This heterogeneity hinders seamless communication and knowledge exchange across IoT systems, necessitating solutions that enhance both system efficiency and cross-platform interoperability [11].

To address these challenges, researchers have introduced the Semantic Internet of Things (SIoT)—a promising framework built on the principles of the Semantic Web to address data heterogeneity and enable semantic interoperability. SIoT enriches IoT data by assigning meaning through standardized semantic structures. These structures describe IoT data, its context, and its relationships, facilitating integration and usability across diverse IoT applications [12]. At the core of SIoT lies ontology, which serves as the backbone for organizing and representing knowledge in a machine-readable format. Ontologies enable the consistent representation and sharing of IoT data. However, SIoT faces significant challenges related to ontology management. A key issue is the reliance of many existing solutions on specific, often rigid ontologies. This has led to a proliferation of conflicting or redundant definitions, making it difficult to ensure interoperability and scalability across diverse systems. The lack of a unified or standardized approach to ontology development and integration further exacerbates these challenges, hindering the seamless exchange of data and knowledge in the SIoT ecosystem. For instance, according to Linked Open Vocabularies for IoT (LOV4IoT), over 400 ontology-based vocabularies have been created [13].

To overcome these limitations, SIoT systems must prioritize the reuse of standardized ontologies. By adopting consistent and widely accepted frameworks, these systems can ensure a more accurate representation of the IoT domain, thereby enhancing data integration and interoperability across diverse platforms. Researchers have proposed several standard ontologies to address interoperability challenges and provide a unified structure for IoT data. However, lot of them have emphasized that these ontologies must be carefully evaluated and adapted to the specific requirements of different IoT applications to ensure their effectiveness and scalability. For example, the SSN ontology [14] was designed with a broad scope encompassing IoT and sensor applications. It provides detailed models of sensors, their characteristics, and measurements, but includes only basic actuator descriptions. It offers medium modularity and can be integrated with other ontologies. Similarly, the oneM2M ontology [15] focuses on M2M (Machine-to-Machine) IoT environments. It provides sensor and device constructs adapted to that context, includes actuator modelling suited to M2M infrastructures, and offers medium modularity with alignment capabilities toward SAREF and other ontologies. Finally, the SAREF ontology, proposed by ETSI [16], targets the IoT and smart building domains. It models both sensors and devices (including actuators) and stands out for its high modularity and explicit alignment with oneM2M and other frameworks. Together, these ontologies offer complementary strengths in domain coverage, modelling depth, alignment potential, and modular design, providing SIoT with a unified yet adaptable semantic foundation for enhanced interoperability and data integration.

While standardized ontologies such as SAREF, SSN, and oneM2M are fundamental for achieving semantic interoperability through consistent data models, their widespread adoption in unifying heterogeneous IoT data also contributes to the volume and scalability challenges. As devices generate vast amounts of data structured according to these ontologies, the resulting RDF datasets grow exponentially. This huge scale poses significant problems for traditional data management techniques. Specifically, the prevailing approach to storing and querying this semantic data relies on RDF triple stores alongside the SPARQL query language. However, as documented in literature and confirmed by our own baseline experiments, these triple store solutions often struggle to maintain acceptable query performance when dealing with the billions of triples characteristic of large-scale IoT deployments. The overhead associated with complex SPARQL query processing and indexing limitations within these stores creates critical bottlenecks, hindering real-time analysis and efficient data retrieval. These limitations necessitate the exploration of alternative architectures and indexing mechanisms beyond standard triple stores, prompting a closer inspection of specific RDF storage approaches and broader SIoT frameworks, as depicted in Table 1 and Table 2.

The Resource Description Framework (RDF) is a key component of SIoT, enabling the representation, querying, and retrieval of data within the ecosystem. Due to its simplicity, adaptability, and reusability, RDF has become a widely adopted standard across various sectors. Consequently, large RDF datasets have emerged, spanning diverse domains such as DBpedia [17], DBLP [18], Bio2RDF [19], and others [20]. These datasets contain billions of RDF triples, with some distributed across multiple files to enhance querying efficiency, as seen with Lehigh University Benchmark (LUBM )[21].

However, managing these massive RDF datasets presents significant challenges related to both storage and retrieval. To address these issues, researchers have proposed several RDF storage solutions, including triple tables [22], binary tables [23] and property tables [24]. These approaches are implemented in platforms such as Apache Jena [25], Blazegraph [26] and GraphDB [27], all of which utilize a triple store model and support SPARQL for querying RDF data across diverse sources.

While these established RDF storage solutions and platforms provide foundational support for semantic data, their architectural choices—particularly their reliance on the triple store model and SPARQL querying—directly impact the scalability and retrieval performance of SIoT systems built upon them. As discussed in the following section, many existing SIoT frameworks—such as M3, SEDIA, and SSNT —adopt these conventional triple store backends. Consequently, despite their advances in semantic interoperability, they often inherit the inherent limitations of these storage mechanisms. Specifically, the overhead associated with processing complex SPARQL queries across potentially billions of triples—combined with indexing strategies optimized for graph patterns rather than the vector-based similarity searches—leads to significant performance degradation at scale (as detailed for triple stores in Table 2 and demonstrated in our baseline comparisons in Section 6). This widespread reliance on architectures that are suboptimal for large-scale, high-speed retrieval serves as a key motivation for exploring alternative approaches which decouples from the traditional triple store model to achieve improved performance.

Data storage plays a critical role in enabling informed decision-making, preserving historical records and collective memory, and facilitating analysis and prediction. In the IoT domain, where vast numbers of devices generate continuous streams of data, effective storage becomes even more essential. This need drives the adoption of various technologies, including cloud computing, edge computing, and hybrid approaches that combine both [28]. SIoT extends IoT by providing a framework for processing data from a wide range of devices and organizing the results into RDF datasets. Several frameworks have been proposed to leverage semantic technologies within IoT. While these frameworks enhance interoperability and semantic understanding, they often face limitations—particularly in efficiently managing large-scale RDF data and addressing inherent data uncertainty. Table 1 presents a comparative summary of representative SIoT frameworks, outlining their methodologies, the limitations addressed by our work, and the improvements introduced by our proposed framework. Beyond individual frameworks, the choice of underlying technologies for storage, classification, and indexing significantly affects performance and suitability in SIoT applications. Table 2 compares common alternatives in these areas.

Table 1: Comparative analysis of representative SIoT frameworks

| **Framework** | **Methodology Key Aspects** | **Key Limitations Addressed by Our Work** | **How Proposed Framework Addresses Limitations** |
|---|---|---|---|
| **VICINITY [29]** | Focus on interoperability (Web of Things), Semantic descriptions, Cloud-centric | Less explicit focus on massive RDF storage/retrieval optimization; Doesn't inherently handle data uncertainty. | Explicitly targets RDF retrieval scalability via custom indexing; Integrates fuzzy logic for nuanced data classification under uncertainty. |
| **BIG-IoT [30]** | Semantic API for interoperability, Uses schema.org, Cloud-based platform | Primarily targets API-level interoperability; Potential scalability bottlenecks if underlying storage isn't optimized for massive RDF query load. | Multi-layered architecture distributes load; Fog layer with custom indexing bypasses typical triple-store query overhead for specific retrieval tasks. |
| **M3 Framework [31]** | Fog/Cloud architecture, M3 Ontology, Uses Jena (Triple Store) for RDF storage | Reliance on standard triple stores (Jena) may lead to known scalability issues and SPARQL query overhead at large scale. | Avoids triple store bottlenecks by using a file-based RDF representation coupled with highly efficient, category-specific binary tree indexing for retrieval. |
| **WooD [32]** | Web Objects concept, Knowledge base storage (details sparse), Aimed at interoperability | Lack of implementation details; Likely relies on traditional knowledge base/triple store querying, risking retrieval delays. | Provides a concrete, implemented architecture with optimized indexing (binary tree) demonstrated to significantly outperform triple store retrieval times. |
| **SEDIA [33]** | Smart City focus, Reuses SSN, IoT-Lite, GeoSPARQL ontologies, Relies on RDF triple stores | Continued dependency on triple stores may pose scalability /retrieval time challenges; Less emphasis on fuzzy classification for nuanced states. | Implements custom binary tree indexing tailored for fast retrieval, bypassing SPARQL overhead; Integrates fuzzy logic for improved classification before indexing. |
| **SSNT [34]** | Lightweight ontologies (IoT-Lite, SSN), Uses | Relies on triple store (GraphDB), facing potential retrieval delays at | Provides experimental validation showing superior scalability and |

| | | | |
|---|---|---|---|
| | GraphDB (Triple Store) for cloud storage | scale; Lacks real-world evaluation/deployment details. | retrieval speed compared to triple stores; Evaluated with real-world derived data. |
| **NoSQL databases [35]** | Stores RDF data using NoSQL models (e.g., document, key-value, column-family). Leverages inherent NoSQL scalability and schema flexibility. | Potential loss of semantic expressiveness compared to native RDF stores. SPARQL querying often inefficient or requires complex mapping. Standard NoSQL indexes not typically optimized for semantic/vector similarity search. | Maintains RDF semantics via file representation & SAREF. Uses optimized binary tree indexing tailored for vector similarity (score + measurements), bypassing SPARQL overhead for targeted retrieval. |
| **Proposed Framework** | Edge (SAREF/RDF standardization) -> Fog (Fuzzy Classification + Binary Tree Indexing) -> Cloud (Storage); Focus on RDF scalability & retrieval speed | Addresses: Scalability limits of triple stores, SPARQL overhead, rigid classification of uncertain data. | Method: Multi-layer processing, Fuzzy logic for nuanced classification, Custom binary tree indexing optimized for categorized RDF file retrieval based on feature vectors (score + measurements). |

Table 2: Comparison of technologies for SIoT data management components

| **Category** | **Technology Option** | **Suitability for SIoT Data (Heterogeneity, Volume, Velocity)** | **Scalability** | **Query Performance (Type)** | **Handling Uncertainty** | **Semantic Support** | **Strengths** | **Weaknesses** |
|---|---|---|---|---|---|---|---|---|
| **Storage Architecture** | Centralized Cloud | High Volume, Moderate Velocity | High | Depends on DB; Can be bottleneck | Indirect | High (via DB) | Flexibility, Central management | Latency, Bandwidth costs, Single point of failure |
| | Edge/Fog Hybrid (Our Approach) | High Heterogeneity, Volume, Velocity | High | Optimized at Fog (Fast Retrieval) | Direct (Fog Logic) | High (Edge/ Fog) | Low latency (Edge), Reduced network load, Distributed processing/ storage | Architectural complexity, Synchronization challenges |
| | Triple Stores (Jena, Blazegraph) | High Heterogeneity, Volume | Medium Low | Good (SPARQL - pattern matching), Poor (Massive scale) | Low | Very High | Native RDF support, SPARQL standard | Scalability limits, Complex query overhead, Poor for vector similarity search |
| | Graph Databases (Neo4j) | Moderate Heterogeneity, High Volume | High | Excellent (Traversal), Moderate (Pattern), Poor (RDF Native) | Low | Mode-rate | Relationship focus, Scalability | Non-native RDF, Different query language (Cypher), Less ideal for pure semantic web integration |
| | NoSQL (Document/KV) | High Heterogeneity, Volume, Velocity | Very High | Varies (Key lookup fast, Complex queries slow) | Low | Low Mode-rate | Scalability, Flexibility | Reduced semantic expressiveness, Inconsistent query performance |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **Clustering/ Classification** | Crisp Thresholds (e.g., Std MEWS) | Simple Data | N/A | N/A | Poor | N/A | Simple, Fast | Prone to misclassification at boundaries, doesn't model vagueness |
| | K-Means, DBSCAN | Moderate Volume | Mode-rate | N/A | Poor | N/A | Unsupervised discovery, Handles certain cluster shapes | Needs parameters (k, eps), Sensitive to noise/shape, not ideal for predefined categories or fuzzy boundaries |
| | Fuzzy Logic (Our Choice) | Heterogene ous, Uncertain Data | N/A | N/A | Very High | N/A | Handles vagueness/un certainty, Interpretable rules, Nuanced output | Needs domain knowledge (rules/MFs), Can be computationally intensive |
| **Indexing Mechanism** | No Index (Linear Scan) | Low Volume | Poor | Poor (O(n)) | N/A | N/A | Simple | Inefficient for large datasets |
| | Triple Store Indexes (SPO, etc.) | High Volume (but struggles at massive scale) | Medium Low | Good (Specific SPARQL patterns), Poor (General/Simi larity) | N/A | High | Optimized for SPARQL | Tied to triple store limitations, not for vector similarity |
| | B-Trees / B+ Trees | Structured Data | High | Excellent (1D Range), Poor (Multi-dim Similarity) | N/A | Low | Standard, Efficient for range queries | Not directly suited for multi-dimensional similarity search |
| | R-Trees / QuadTrees | Spatial / Multi-dimensional Data | High | Good (Spatial Range/Proxim ity), Moderate (Similarity) | N/A | Low | Efficient multi-dimensional spatial queries | Can be complex, overhead for non-spatial similarity, Potential overkill |
| | Graph DB Indexes | Graph Data | High | Excellent (Node/Relatio nship lookup) | N/A | Mode-rate | Optimized for graph traversal | Not for RDF vector similarity |
| | Custom Binary Tree (Our Choice) | Categorized Vector Data | High (per tree) | Good (Avg O(log n) Similarity Search) | N/A | Mode-rate (via vector) | Tailored for vector similarity, Synergizes with classification | Needs balancing consideration, Worst-case O(n) if unbalanced |

## 4 Proposed framework

Despite significant advancements in the Semantic Internet of Things (SIoT), critical challenges persist—particularly in efficiently managing the exponential growth of RDF data generated by the rapidly increasing number of IoT devices. Addressing this challenge requires a more innovative and scalable approach to data storage, organization, and retrieval to meet the growing demands of modern, data-driven applications. A promising solution lies in a layered architectural framework that seamlessly integrates edge, fog, and cloud computing components. This approach addresses critical challenges such as real-time data processing, efficient clustering, robust indexing, and scalable storage—ensuring optimized performance and resource utilization across all system layers.

Clustering is an effective technique for grouping similar data while maintaining clear separation between different elements. When addressing overlapping or ambiguous data, integrating fuzzy logic into the clustering process significantly enhances its effectiveness—improving categorization accuracy and enabling better handling of complex datasets. Furthermore, indexing techniques can significantly improve search efficiency by organizing data into hierarchical structures [36]. This approach reduces the search space and accelerates query-based searches, enabling faster and more precise data retrieval. The practical deployment of such SIoT frameworks in real-world domains is essential for evaluating their effectiveness, identifying tangible benefits, and fostering broader adoption and innovation. These objectives form the focus of the following sections, where we demonstrate and discuss the capabilities of our proposed framework.

## 4.1 System architecture overview

Timely processing of IoT data, along with efficient storage and RDF retrieval, is critical. Cloud computing, with its bandwidth constraints, struggles to meet these real-time requirements. To address this limitation, an architecture that ensures low-latency processing and fast data access is essential for delivering responsive and efficient services. To meet these demands, we propose a robust edge-fog-cloud architecture, as illustrated in Figure 1. This architecture is structured into four layers:

1. Device Layer: Responsible for generating IoT data.
2. Edge Layer: Processes data near the devices to reduce latency and alleviate network congestion.
3. Fog Layer: Distributes storage and computation tasks across intermediate nodes.
4. Cloud Layer: Provides centralized storage and long-term data management.

This multi-layered design optimizes data handling across the IoT ecosystem by balancing speed, efficiency, and scalability to accommodate the growing demands of modern IoT applications—including those involving a large number of connected devices. The first layer, the IoT Device Layer, encompasses all devices with data collection capabilities —such as sensors, wearables, and

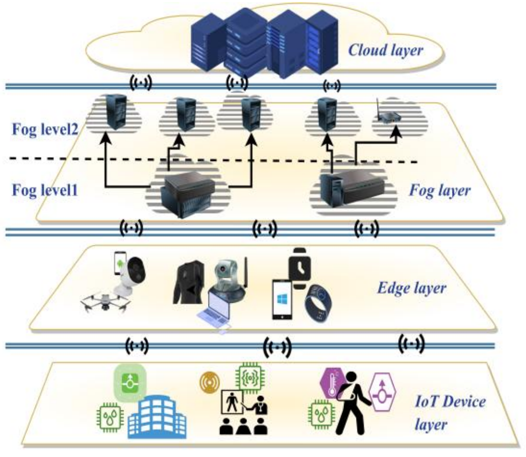


Figure 1: Edge-Fog-Cloud layer architecture for efficient IoT data processing and management

smart devices—and communication technologies like Wi-Fi, Bluetooth, and others used for data transmission. The Edge Layer includes resources located near these devices, such as gateways, routers, and local servers. Devices like smartwatches and desktops can also function as edge nodes. This layer is crucial for ensuring scalability across multiple devices, as it distributes initial processing load.

The Edge Layer processes data received from sensors and performs initial semantic enrichment. It incorporates a semantic engine that performs reasoning over the data, enabling more insightful analysis at this early stage. By leveraging the ontology, the semantic engine annotates the sensors' measurement values and generates results in RDF format. This enables more accurate interpretation and categorization of sensor data, laying the foundation for decision-making and further processing in subsequent layers of the system. However, due to the limited storage capacity for RDF files at the edge, the data must be transferred to the next layer—the Fog Layer. The Fog Layer acts as an intermediary, comprising more powerful fog nodes (e.g., servers, routers) located regionally or within specific facilities such as hospitals. It bridges edge devices and the central cloud, further alleviating network load and enhancing security. This layer aggregates data from multiple edge nodes within its domain. The Fog Layer operates through two key phases that are central to the efficiency of our framework:

1. Clustering Phase: Utilizes a fuzzy logic algorithm (Algorithm 1) to classify incoming RDF files from the Edge Layer into meaningful categories based on content.
2. Indexing Phase: Constructs category-specific binary trees (Algorithm 2) to organize and index the classified RDF data vectors, enabling highly efficient similarity-based retrieval (Algorithm 3).

By combining these phases, the Fog Layer efficiently manages data while minimizing network overhead—ensuring faster and more secure data handling within the IoT ecosystem. Figure 2 presents a visual representation of the complete data processing pipeline. This flowchart illustrates how the core algorithmic components, detailed in subsequent sections, interact within the layered architecture.

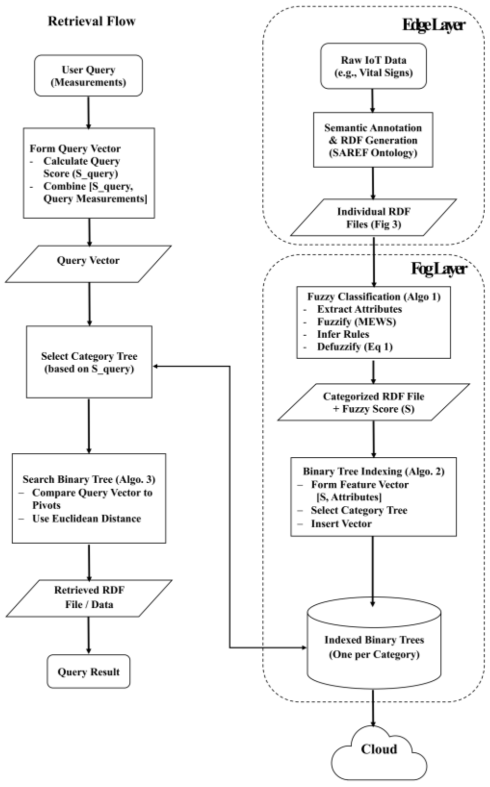

Retrieval Flow
User Query (Measurements)
Form Query Vector
- Calculate Query Score (S_query)
- Combine [S_query, Query Measurements]
Query Vector
Select Category Tree (based on S_query)
Search Binary Tree (Algo. 3)
– Compare Query Vector to Pivots
– Use Euclidean Distance
Retrieved RDF File / Data
Query Result
Edge Layer
Raw IoT Data (e.g., Vital Signs)
Semantic Annotation & RDF Generation (SAREF Ontology)
Individual RDF Files (Fig 3)
Fog Layer
Fuzzy Classification (Algo 1)
- Extract Attributes
- Fuzzify (MEWS)
- Infer Rules
- Defuzzify (Eq 1)
Categorized RDF File + Fuzzy Score (S)
Binary Tree Indexing (Algo. 2)
– Form Feature Vector [S, Attributes]
– Select Category Tree
– Insert Vector
Indexed Binary Trees (One per Category)
Cloud

Figure 2: Data Processing and retrieval workflow

The proposed architecture is designed to support dynamic environments in which devices frequently join and leave the network. Device discovery, authentication, and connection management are primarily handled at the Edge Layer, utilizing standard IoT protocols and, potentially, an IoT platform's management plane. When a new device joins, the Edge Layer registers it and incorporates its data into the processing pipeline. Subsequent fuzzy classification and binary tree indexing in the Fog Layer operate on the data's characteristics (i.e., the feature vector derived from RDF), rather than relying on persistent device connections. If a device disconnects, the Edge Layer simply stops receiving its data, while previously generated and indexed information remains available. While optimizing specific protocols for high-frequency join/leave scenarios remains an area for future work, the decoupled nature of the data flow ensures that core data processing and indexing mechanisms remain robust and scalable in dynamic environments.

## 4.2 Framework modules: core components of the architecture

Internet of Things (IoT) devices typically generate and transmit raw data over the Internet. The inconsistency between devices makes processing and exchanging this data highly challenging. Semantic Web technologies help bridge the gap between heterogeneous devices and enable semantic interoperability. The Semantic Web is widely used for representing data; it supports advanced reasoning and decision-making and facilitates the integration of data from multiple sources. RDF serves as the foundational data model for the Semantic Web, allowing seamless integration and sharing of data across diverse applications and domains. It also provides a suitable model for storing any semantic web data. From IoT devices that collect data, to its processing and storage in RDF format, and finally to timely data retrieval, we propose an approach that combines Semantic Web technologies with data structuring mechanisms to enhance data organization and processing.

The proposed framework, depicted in Figure 3, is designed for IoT environments. It supports the management of data heterogeneity and stores the resulting data in RDF format. The framework relies on clustering and indexing of RDF files to enable rapid data retrieval. The approach consists of four main components:

### 4.2.1 Semantic representation

The first layer of the proposed approach addresses the challenge of IoT data heterogeneity. It leverages Semantic Web technologies and comprises a set of integrated modules. At its core lies the knowledge base module, which provides a formal structure to facilitate the sharing and reuse of knowledge. This module includes both an ontology and additional domain knowledge. The ontology defines a vocabulary for representing data collected from sensors by specifying relevant concepts and the relationships between them. It serves as the backbone of the Semantic Web, enabling semantic consistency and meaning-sharing across different IoT applications. While many existing methods support the creation of custom ontologies tailored to the IoT domain, reusing well-established ontologies significantly enhances collaboration and data exchange across heterogeneous systems.

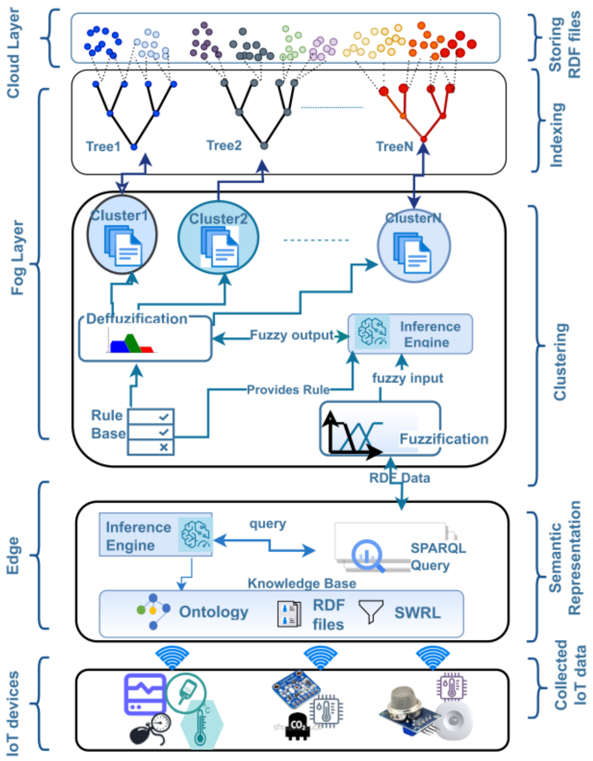


Figure 3: Core elements and structural components of the architecture

In particular, the adoption of the standardized SAREF ontology streamlines the alignment and sharing of ontologies, promotes interoperability, and ensures cohesive integration across diverse IoT platforms. Beyond the ontology itself, the knowledge base incorporates essential supplementary information about the IoT domain, including domain objectives, contextual information, historical measurement data, and more. This additional layer of knowledge enriches the ontology, offering a more holistic and accurate representation of the domain, thereby enabling deeper insights and improving decision-making and interoperability. The knowledge base module also integrates SWRL (Semantic Web Rule Language), which allows the expression of complex logic in the form of rules. These rules can be used to infer new, hidden knowledge from data stored in RDF format. This capability enables the discovery of implicit relationships and insights that are not explicitly stated in the original dataset. In addition to the knowledge base, the architecture includes an inference engine and a SPARQL engine. The inference engine reasons over existing data to infer new facts, while the SPARQL engine enables querying and retrieval of source data. To facilitate the use of this knowledge, preserve its semantic meaning, and enable interoperability across different applications, the information is stored in RDF file format. An RDF file is a machine-readable document that uses the RDF language to structure IoT data semantically.

For example, in the IoT domain, RDF files may contain historical sensor measurements, environmental information, and other contextual data. Since tracking the history of IoT data is highly beneficial, storing this information in RDF format simplifies its processing, analysis, and prediction. However, as the semantic component processes increasing volumes of data, the size of RDF files grows substantially, leading to challenges related to storage capacity and retrieval efficiency. To mitigate these issues, it is essential to introduce a module that structures and optimizes the data effectively. Clustering offers a promising solution by organizing data into manageable groups before the indexing process. This approach reduces the volume of data to be indexed, improving system efficiency and scalability.

While the current implementation leverages SAREF for standardization, the framework's modular architecture allows for adaptation to environments that already utilize other standard ontologies such as SSN or oneM2M. The semantic representation module, which operates primarily at the Edge Layer, is conceptually decoupled from the Fog Layer's fuzzy classification and indexing logic—both of which operate on feature vectors extracted from RDF data. Integration can be achieved either by developing ontology mapping layers (using OWL, RDFS, or SKOS) to translate SSN or oneM2M data into the SAREF structure expected downstream, or by directly adapting or replacing the semantic module to generate RDF based on the target ontology (e.g., SSN or oneM2M). While the latter approach would require reimplementing the RDF generation logic, it could offer tighter integration with existing systems.

Crucially, as long as the necessary numerical features can be reliably extracted from the resulting RDF—regardless of the specific ontology vocabulary used—for input into the fuzzy classifier and indexing vector, the core logic of the Fog Layer modules (Algorithms 1, 2, and 3) remains applicable with minimal modification. Therefore, adapting the framework primarily involves adjustments to the semantic representation layer and does not require a complete repair of the multi-layered architecture or its core data processing techniques.

Fragmentation often arises from heterogeneous data sources employing different data models. By using the SAREF ontology for semantic annotation directly at the Edge Layer, the framework establishes a consistent RDF structure for all incoming data before it is distributed. Each generated RDF file represents a coherent snapshot of related measurements, logically linked via the ontology as illustrated in Figure 3. This early standardization minimizes structural fragmentation from the outset. The Clustering Module groups these semantically coherent RDF files into categories based on their content. As a result, files representing similar states or events are logically grouped together, facilitating more targeted and efficient querying. Although data may still reside in numerous individual files, the Binary Tree Indexing approach enables efficient retrieval based on the content vector. A query for a specific state retrieves the most similar file(s) directly via the index, eliminating the need to manually reconstruct fragmented information across unrelated files or perform complex SPARQL joins in a potentially fragmented triple store. While the framework does not physically merge files, the combination of logical categorization and content-based indexing effectively addresses the retrieval challenges associated with fragmentation.

### 4.2.2 Clustering module

Operating primarily within the Fog Layer, as described in Section 4.1, the Clustering Module processes the RDF files produced by the semantic layer at the Edge. Following the semantic representation phase detailed in Section 4.2.1—where heterogeneous IoT data is standardized into a consistent RDF format using the ontology—the framework encounters the subsequent challenge of efficiently managing and retrieving information from the potentially vast volume of generated RDF files. While semantic standardization addresses issues of interoperability and shared meaning, it does not inherently resolve the scalability challenges associated with searching through large numbers of individual files, particularly when rapid access is required. Therefore, the next critical step involves organizing these semantically enriched RDF files based on their content to facilitate optimized access. This task is performed by the Clustering Module, which classifies the RDF files into meaningful categories based on extracted features and computed scores. By doing so, the module effectively structures the data, significantly reduces the search space, and lays the foundation for the efficient indexing and retrieval mechanisms described in subsequent sections.

To address the challenges posed by massive data volumes and to improve retrieval efficiency, this module groups RDF files representing similar states into distinct categories. This categorization structures the data, reduces the search space for subsequent retrieval, and accelerates query performance by ensuring that related data is indexed together. The approach involves constructing a separate index—specifically, a binary tree—for each category. This significantly speeds up the search process, as each index contains only RDF files belonging to its respective group. Moreover, by employing fuzzy logic, the module enhances the framework's adaptability to dynamic environments characterized by uncertainty or gradually evolving data, ultimately supporting more robust and responsive data-driven decision-making.

Given that IoT data is often imprecise and vague, attempting to classify it using rigid thresholds can result in inaccurate outcomes. Fuzzy logic emerges as a highly suitable solution, specifically designed to handle the uncertainty and ambiguity inherent in such data. It enables reasoning under uncertainty by using degrees of membership rather than absolute categorizations. By mirroring human decision-making, fuzzy logic effectively models overlapping concepts and gradual transitions between data states. The decision to employ fuzzy logic is fundamentally driven by its superior capability to manage the vagueness and imprecision characteristic of real-world IoT data. Unlike traditional methods based on crisp thresholds which struggle with ambiguity at category boundaries and risk misclassifying borderline cases, fuzzy logic explicitly accounts for this imprecision. Through the use of membership functions, fuzzy logic allows a data point to possess partial membership in multiple linguistic categories.

The ability to represent partial truths produces a more nuanced output score (S), which facilitates more accurate and relevant classification into predefined categories. This enhanced level of precision is especially valuable in high-stakes domains such as healthcare and was qualitatively supported by expert review in our study. Furthermore, the rule-based structure inherent to many fuzzy systems offers a level of interpretability often lacking in complex black-box models. Alternative approaches were considered less appropriate for this task: crisp thresholds are inherently limited in their ability to represent ambiguity, while standard unsupervised clustering algorithms (e.g., K-Means, DBSCAN) are primarily intended for discovering unknown data groupings rather than performing fine-grained classification into predefined, semantically meaningful categories with inherently fuzzy boundaries.

It is important to note that within the context of our target healthcare application, this module performs fuzzy classification rather than unsupervised clustering. The objective is to assign each RDF file to one of the predefined categories, using fuzzy logic to enable a more nuanced and robust classification than that achieved through standard classification methods. The fuzzy system employs several standard components to perform this classification. The process begins by extracting relevant features—termed Attributes—from each RDF file f. As these files represent IoT measurements structured according to the SAREF ontology, the Attributes typically consist of specific numerical values such as pulse rate or temperature in health context.

Following extraction, the fuzzifier module converts these numerical attributes into fuzzy sets. It applies predefined membership functions, such as triangular or trapezoidal shapes, with boundaries derived from established guidelines. Each numerical value is assigned a degree of membership between 0 and 1 for the corresponding linguistic terms, such as "Normal" or "High1" in the case of heart rate. These terms and their associated membership functions are domain-specific and grounded in medical standards such as the MEWS system for vital signs. The output of this stage is a fuzzy representation of the input data. (e.g.: A heart rate of 115 may fuzzify to {Normal: 0.0, High1: 0.4, High2: 0.8}).

The core fuzzy inference component then utilizes a set of fuzzy rules—defined by domain experts—to evaluate the fuzzified inputs. These rules map combinations of fuzzy input values to output fuzzy sets that represent the target classification categories, such as risk levels. A representative rule might be: *IF Pulse is High1 AND OxygenSaturation is Low1 THEN Risk is Moderate*. The inference engine evaluates all applicable rules for the given input, aggregates the results, and produces a final fuzzy output set that indicates the file's degree of compatibility with each risk category. Finally, to convert the fuzzy output into a single crisp score, the defuzzification module applies the Centroid method, defined as:

$$S = \frac{\sum_{i=1}^{n} x_i \cdot \mu(x_i)}{\sum_{i=1}^{n} \mu(x_i)} \quad (1)$$

Where:

- $S$ is the crisp output score (denoted as $\text{Score}_i$),
  $x_i$ represents a point in the output domain (e.g., a specific risk level or severity index),
- $\mu(x_i)$ is the membership degree of the aggregated fuzzy output set at point $x_i$,
- $n$ is the total number of discrete points considered within the output domain.

This resulting score $S$ quantifies the degree of compatibility or relevance of a given RDF file $f$ to a specific target class $C_{\ell_i}$. It is this crisp value that is then passed forward to the classification and indexing modules within the SIoT architecture.

Algorithm 1 outlines the process for classifying RDF files into categories using the fuzzy logic engine. This algorithm leverages the computed fuzzy scores to assign each data point (i.e., RDF file) to the most appropriate predefined category. By calculating the compatibility score of each data point with all available categories, the algorithm effectively handles ambiguity and overlapping concepts—particularly useful in scenarios where clear boundaries between states are not well-defined. Based on these scores, each file is assigned to the single best-fitting category, thereby facilitating optimized data organization for subsequent indexing and retrieval.

**Algorithm 1: Clustering with Fuzzy Logic Engine**
**Input**:

```
• F = {f1, f2, ..., fm}  // Dataset of RDF files
• Cl = {Cl1, Cl2, ..., Cln}  // Initial set of clusters
• FuzzyLogicEngine // Module to compute membership scores
• ε  // Convergence threshold for early stopping
Output:
• Clusters with assigned RDF files
Steps:
1. Initialize Cl ← DefineInitialClusters(F)  // Use domain-specific rules
2. T ← CalculateInitialThreshold(F, Cl)  // Compute initial threshold based on average membership scores
3. Repeat:
4.   AssignmentsChanged ← False  // Track changes in cluster assignments
5.   For each file f ∈ F:
6.     Attributes ← ExtractAttributes(f)// Extract features from f
7.     Scores ← []  // Initialize empty list for scores
8.     For each cluster Cli ∈ Cl:
9.       Scorei ← FuzzyLogicEngine.CalculateScore(Attributes, Cli)
10.      Append (Cli, Scorei) to Scores
11.    BestCluster, BestScore ← max(Scores)  // Find cluster with highest score
12.    If BestScore > T:
13.      Assign f to BestCluster
14.      If f was previously assigned to a different cluster:
15.        AssignmentsChanged ← True
16.    Else:
17.      createNewCluster(f)// Initialize new cluster with f as centroid
                            Append new cluster to Cl
18.      AssignmentsChanged ← True
19.  UpdateClusters(Cl)  // Adjust centroids using weighted averages of assigned files
20.  T ← AdjustThreshold(F, Cl)  // Recalculate threshold based on updated clusters
21.  Convergence ← CheckConvergence(AssignmentsChanged, ε)  // Stop if changes are below ε
22. Until Convergence = True or AssignmentsChanged = False
23. Return Cl  // Output clusters with assigned RDF files
```

In the initialization phase of Algorithm 1, the set of classification categories $C_l$ is defined. Although generic clustering techniques can be employed, the proposed framework favours a domain-specific approach in which these categories are predefined based on context-relevant semantic criteria. This ensures alignment with real-world conditions and facilitates the interpretation of results. The classification process requires the calculation of a threshold $T$, which represents the minimum score of confidence required to assign a given data file to a particular category. When a file's highest fuzzy compatibility score falls below this threshold, the system interprets it as indicative of uncertainty or weak association. In such situations, instead of generating a new category, the file may be allocated to a designated "uncertain" group or flagged for manual review, particularly in domains where categories are fixed and semantically meaningful.

The computed fuzzy score is the primary determinant for identifying the most appropriate classification. Unlike soft clustering methods such as Fuzzy C-Means, which assign partial membership values across multiple clusters and refine them iteratively, the approach adopted here aims for a crisp categorization. Each file is ultimately assigned to the most compatible predefined category, based on its highest fuzzy score. This strategy is especially advantageous in scenarios that require clear organizational structure and efficient retrieval, while still accommodating uncertainty and overlapping semantic boundaries through the use of fuzzy logic.

A complexity analysis of Algorithm 1 reveals its computational cost as follows: Let $m$ represent the number of RDF files, $n$ the number of predefined categories, $I$ the number of iterations (potentially just one for simple classification), and $d$ the feature dimensionality. Let $C_{\text{extract}}$ be the cost of attribute extraction per file, and $C_{\text{fuzzy}}$ the cost of fuzzy scoring per file per category (dependent on fuzzy rule complexity). The initialization step, based on domain-specific rules, typically has a complexity of $O(n)$. The overall time complexity is dominated by the loop that processes each file against every category, resulting in a total complexity of approximately:

$$O\left(I \cdot m \cdot \left(C_{\text{extract}} + n \cdot C_{\text{fuzzy}}\right)\right) \tag{2}$$

This expression highlights the algorithm's dependence on the dataset size $m$, the number of target categories $n$, and the computational costs associated with RDF feature extraction and fuzzy rule evaluation.

### 4.2.3 Indexation layer

Following the classification step—also performed within the Fog Layer—the Indexation Layer, operating in the same environment, employs indexing as a powerful and widely adopted technique to enable rapid access to large datasets. In this work, we utilize a binary tree structure to index RDF files within each category determined by the fuzzy classification module described in Section 4.2.2. A separate binary tree is constructed for each category; for example, one tree per MEWS risk level in the healthcare scenario. This strategy partitions the data based on its classified state prior to indexing. Each node $N$ within these binary trees may contain references to up to two child nodes (leftChild, rightChild) and is designed to hold up to two pivot points (Pivot1, Pivot2). The basic structure of a node is defined as follows: Node { leftChild: Node | null; rightChild: Node | null; Pivot1: Vector | null; Pivot2: Vector | null; }, where Vector represents the feature vector associated with an RDF file.

The choice of this custom binary tree structure is motivated by the need for efficient retrieval from potentially massive collections of categorized RDF files—a critical requirement in which standard SIoT approaches, relying on triple stores and SPARQL queries, often struggle due to scalability.

In our framework, RDF data is transformed into categorized files, each represented by a multi-dimensional feature vector comprising the fuzzy score and relevant sensor measurements. The binary tree index is specifically optimized for fast similarity-based retrieval of these vectors within their assigned categories. Its primary advantage lies in the hierarchical partitioning of the vector space using Euclidean distance relative to the pivot vectors stored in each node. This structure enables an average-case retrieval complexity potentially approaching $O(\log n)$, where $n$ is the number of files in a category's tree.

This logarithmic complexity represents a significant improvement over linear search methods and avoids the overhead associated with complex SPARQL pattern matching over large triple stores. Moreover, the indexing approach synergizes directly with the preceding fuzzy classification step: building separate trees per category drastically reduces the search space ($n$) for any given query, as the query's fuzzy score immediately directs the search to the appropriate tree.

While alternative multi-dimensional indexing structures exist (e.g., R-trees, QuadTrees), the chosen binary tree—empirically designed with two pivots per node for effective branching—offers a tailored solution for vector similarity comparison with lower implementation complexity for this specific use case. In contrast, traditional B-trees are not suitable for multi-dimensional similarity tasks, and graph databases introduce non-native RDF handling and alternative query paradigms that are not optimized for this type of vector-based retrieval.

The connection between the clustering phase (fuzzy classification) and the indexing phase is critical to the efficiency of the overall framework. The process begins once an RDF file $f$ has been assigned to a specific category $C_l^i$, and its corresponding fuzzy compatibility score $S$ has been computed. This fuzzy score $S$ is primarily used to determine which category-specific binary tree the RDF file should be inserted into. After the appropriate tree is selected, an input feature vector is constructed to represent the RDF file for insertion. To clarify the vector formation process, this multi-dimensional vector explicitly combines the fuzzy score $S$ with the relevant numerical measurement attributes (e.g., vital signs) extracted from the RDF file. A typical structure is:

$$\text{vector} = [S, \text{measurement}_1, \text{measurement}_2, \ldots, \text{measurement}_d]$$

This complete vector interacts with the binary tree structure during both insertion and retrieval operations. The binary tree structure is specifically designed to optimize data retrieval efficiency. Queries navigate the tree by comparing a query vector to the pivot vectors stored at each node. At each internal node, the algorithm calculates the distance to both pivots and proceeds down the branch associated with the closer pivot. This hierarchical search, guided by vector similarity, dramatically reduces the search space compared to conventional linear scans. The insertion of a new RDF file's feature vector into its designated category-specific binary tree is handled by Algorithm 2. This algorithm preserves the organization of the tree by placing new vectors based on their similarity to existing pivot vectors, ensuring both balance and retrieval efficiency.

**Algorithm 2: Inserting RDF Files into Binary Tree**

**Input**:

vector: The feature vector for an RDF file, including fuzzy score and measurement values.

N: The root node of the specific binary tree corresponding to the vector's category.

**Output**:

Updated binary tree root node.

**Steps**:

```
1. Function InsertVector(N, vector):
2.  If N is null: // Tree/subtree is empty
3.      Return Creationofnode(vector) // Create root or new leaf
4.  // --- Pivot Assignment ---
5.  If N.Pivot1 is null:
6.      N.Pivot1 = vector // Assign vector as the first pivot
7.      Return N
8.  Else if N.Pivot2 is null:
9.      N.Pivot2 = vector // Assign vector as the second pivot
10.     Return N
11. // --- Distance Computation & Child Node Direction ---
12. Else: // Both pivots exist, decide which subtree to traverse
13.     D1 = EuclideanDistance(vector, N.Pivot1) // Use Eq. 3
14.     D2 = EuclideanDistance(vector, N.Pivot2) // Use Eq. 3
15.     If D1 < D2: // Closer to Pivot1
16.         If N.leftChild is null:
17.             N.leftChild = Creationofnode(vector)
18.         Else:
19.   N.leftChild=InsertVector(N.leftChild, vector) //Recurse left
20.     Else: // Closer to Pivot2 (or equal)
21.         If N.rightChild is null:
22.             N.rightChild = Creationofnode(vector)
23.         Else:
24.  N.rightChild=InsertVector(N.rightChild, vector)//Recurse right
25.  Return N // Return the (potentially updated) current node N
26.Function Creationofnode(vector):
27. // Creates a new node instance based on the defined structure
28. N_new = new Node()
29. N_new.Pivot1 = vector // The input vector becomes the first pivot of the new node
30. N_new.Pivot2 = null
31. N_new.leftChild = null
32. N_new.rightChild = null
33. Return N_new.
```

Expanding on the insertion logic, the process begins at the root node $N$ of the relevant binary tree. If node $N$ is null—indicating either an empty tree or an uninitialized subtree—the CreationOfNode function is invoked. This function instantiates a new Node object based on the structure previously defined, assigns the entire input vector as Pivot1 for the newly created node, and initializes all other fields to null. If the current node $N$ is not null but contains fewer than two pivots, the input vector is assigned to the next available pivot slot (Pivot1 or Pivot2). It is important to clarify that the entire multi-dimensional input vector is used as the pivot; no aggregation or selection of specific values from the vector is performed. When both pivots are already present in node $N$, the algorithm proceeds to calculate the distance between the input vector and each of the node's existing pivot vectors. To ensure clarity on the distance computation involving multi-dimensional vectors, the algorithm employs the standard Euclidean distance metric in a multi-dimensional space.

$$d(\mathrm{v}, \mathrm{p}) = \sqrt{\sum_{i=1}^{n}(v_i - p_i)^2} \qquad (3)$$

Here, $d(v, p)$ denotes the Euclidean distance between the input vector $v$ and a pivot vector $p$, with the sum computed across all $i$ dimensions of the vectors—including the fuzzy score and all associated measurement values. Based on which pivot (Pivot1 or Pivot2) yields the smaller distance (denoted as $D_1$ or $D_2$), the algorithm recursively calls the InsertVector function on the corresponding child node (leftChild or rightChild). This recursive process ensures that similar vectors are grouped within the same subtrees, continuing until the input vector is either inserted as a pivot in an existing node or assigned as Pivot1 in a newly created leaf node. The structural integrity and efficiency of the binary tree are maintained through this recursive insertion strategy, which is guided by Euclidean similarity within the multi-dimensional feature space.

### 4.2.4 Data search and retrieval

The data retrieval process leverages the indexes created and maintained within the Fog Layer. While a user query may originate from an application interfacing with the Cloud Layer or other external systems, the core search mechanism—detailed in Algorithm 3—executes primarily within the Fog Layer, utilizing the locally held binary tree indexes. Efficient data retrieval is essential for extracting value from large-scale IoT datasets. Our approach uses the structured binary trees built in the Indexation Layer (Section 4.2.3) to perform rapid similarity searches for RDF files based on their feature vectors, aiming to optimize both speed and accuracy in retrieving relevant data.

The search process begins with the formulation of a query, typically represented by a set of target measurement values. This input is first processed through the same fuzzy logic engine used during the clustering/classification phase (Section 4.2.2) to compute a query fuzzy score ($S_{\text{query}}$). This score determines which category-specific binary tree should be queried. A query vector ($\text{vector}_{\text{query}}$) is then constructed—analogous to the vectors used for insertion—by combining the query score with the target measurement values:

$$\text{vector}_{\text{query}} = [S_{\text{query}}, \text{query\_measure}_1, \ldots, \text{query\_measure}_d]$$

The search within the selected tree begins at the root node $N$. If $N$ is null, the search terminates with no relevant data found. If $N$ is not null, the algorithm compares the query vector to the pivot vectors stored in the node using standard multi-dimensional Euclidean distance. Regarding mixed value types, both the fuzzy score and the numerical measurements are treated as numerical dimensions within the vector space. The Euclidean distance calculation inherently handles this multi-dimensional comparison:

$$d(v_{\text{query}}, p) = \sqrt{(S_{\text{query}} - S_{\text{pivot}})^2 + \sum(\text{query\_measure}_i - \text{pivot\_measure}_i)^2} \quad (4)$$

Here, $v_{\text{query}}$ is the query vector, $p$ is a pivot vector, $S$ denotes the fuzzy score component, and $\text{measure}_i$ represents the $i$-th measurement value. Each dimension—both score and measurements—is treated equally in the distance computation. No explicit weighting is applied in the standard Euclidean distance formula; however, the relative influence of the fuzzy score versus the measurement values can be implicitly adjusted during the feature scaling phase if desired. In our implementation, standard feature scaling was used.

Algorithm 3 outlines the recursive traversal logic used during data retrieval. At each node $N$, the algorithm calculates the distance $distance_1$ between the query vector and $N$.*Pivot1*. If $N$.*Pivot2* is present, a second distance $distance_2$ is also computed relative to that pivot. The algorithm first checks for an exact match—that is, whether either distance equals zero. If no exact match is found, traversal is guided entirely by the Euclidean distances: if $distance_1 < distance_2$, the algorithm proceeds recursively down the left subtree (closer to *Pivot1*); otherwise, it traverses the right subtree (closer to *Pivot2*). It is important to note that the fuzzy score serves primarily to select the appropriate binary tree during query initialization. Within the selected tree, the search is based solely on geometric proximity in the multi-dimensional feature space, as determined by Euclidean distance. This ensures that the search path consistently moves toward the vectors most similar to the query vector, considering all included features—both the fuzzy score and the measurement values. The recursive process continues until an exact match is identified or a null node (indicating a leaf or end of branch) is reached. This hierarchical search significantly reduces the number of comparisons required, offering a clear performance advantage over traditional linear scanning.

**Algorithm 3: Data Search & Retrieval**

```
Input:
N: A node in the binary tree (currently visited).
  vector_query: The input query vector (score + measurements).
Output:
The retrieved RDF file's representative vector (Pivot) or null.
Procedure SearchRDFFiles(N, vector_query):
1.  If N is null:
2.      Return null  // Reached end of branch, no match found
3.  distance1 ← EuclideanDistance(vector_query, N.Pivot1)
4.  If distance1 == 0:
5.      Return N.Pivot1  // Exact match with Pivot1
6.  If N.Pivot2 is not null:
7.      distance2 ← EuclideanDistance(vector_query, N.Pivot2)
8.      If distance2 == 0:
9.          Return N.Pivot2  // Exact match with Pivot2
10.     If distance1 < distance2:
11.         Return SearchRDFFiles(N.leftChild, vector_query)  //
                    Recurse left
12.     Else:
13.         Return SearchRDFFiles(N.rightChild, vector_query)  //
                    Recurse right
14. Else:
15.     Return SearchRDFFiles(N.leftChild, vector_query)
```

The efficiency of the proposed binary tree indexing approach (Algorithm 3) merits comparison with alternative retrieval methods. In the best-case scenario, the query complexity of our method is approximately $O(d \cdot \log \quad m)$, where $m$ is the number of indexed RDF files and $d$ is the dimensionality of the feature vector, assuming a reasonably balanced tree. However, as with many tree-based structures, the worst-case query complexity can degrade to $O(d \cdot m)$ if the tree becomes highly unbalanced due to non-uniform data insertion patterns. This performance stands in contrast to graph traversal techniques commonly used in RDF triple stores. While such methods can be efficient for specific SPARQL relationship queries using pre-optimized indexes, their worst-case complexity for complex pattern matching can be significantly higher—potentially polynomial—and they are not inherently designed for the type of vector-based similarity search employed in our framework. Standard hash-based indexing offers excellent average-case performance ($O(1)$) for exact-match lookups, but it is inherently unsuitable for distance-based similarity searches without specialized adaptations such as Locality-Sensitive Hashing (LSH), which introduces approximation and may compromise accuracy. Therefore, while we acknowledge the possibility of worst-case $O(d \cdot m)$ performance, the proposed binary tree structure represents a task-specific, efficient solution for retrieving RDF files based on similarity within the multi-dimensional feature space defined by measurement values and fuzzy scores. It outperforms traditional graph traversal or hashing approaches in direct applicability to this retrieval objective.

## 5 Application scenario: IoT-driven enhancements in healthcare

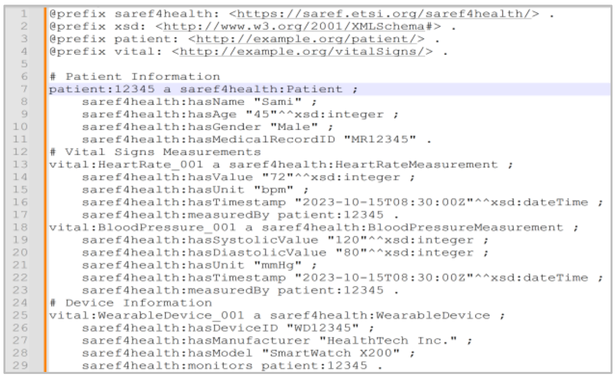
Healthcare is a critical priority for governments and health organizations as they strive to protect populations from disease, enhance diagnostic capabilities, and promote overall well-being. The integration of IoT technologies has transformed this sector, introducing advanced capabilities that improve diagnostic accuracy, reduce costs and delays, and enable continuous patient monitoring. This domain is particularly well-suited for the application of our proposed approach, which can have a significant impact. Timely and accurate data management is essential in healthcare, as it not only has the potential to save lives but also to improve patient outcomes. By enabling more efficient and reliable data processing, our approach supports better clinical decision-making and enhances the overall quality of care. Vital signs—such as heart rate, blood pressure, and oxygen saturation—are essential indicators of a patient's condition [37]. IoT devices enable the real-time collection of these measurements and facilitate immediate transmission to healthcare providers or hospitals. This capability supports early detection of potential health issues and enables continuous monitoring, particularly for patients with chronic conditions. Beyond data collection, IoT technologies rely on robust architectures to process, store, and retrieve this critical information [38]. Paradigms such as cloud computing offer scalable solutions for data storage and processing, while semantic processing addresses data heterogeneity—ensuring that diverse data formats can be effectively integrated. These architectures facilitate the seamless transmission and retrieval of actionable insights, empowering healthcare professionals to analyze patient data efficiently [39]. Our approach aligns with these objectives by introducing a semantic-driven IoT architecture that enables faster and more accurate data processing while ensuring proper categorization and storage. By addressing key challenges such as data heterogeneity and retrieval latency, this framework equips healthcare providers with the tools needed to make informed, timely decisions—ultimately improving the quality of care and patient outcomes.

According to the proposed IoT framework, the healthcare system architecture is structured into multiple layers. The first layer comprises all health-related IoT devices responsible for generating and transmitting patient data. These devices include sensors capable of measuring vital signs—such as wearable devices, smartphones, and other smart technologies—with limited memory and processing capabilities. They enable seamless real-time data processing and analysis, supporting immediate insights and decision-making at the edge of the network. These smart devices utilize Semantic Web technologies to address data heterogeneity, producing RDF files for standardized representation. The third layer consists of devices such as routers and servers, deployed within hospitals and covering broader geographic areas. This layer is subdivided into Fog Level 1, which handles the clustering of RDF files, and Fog Level 2, which performs indexing. The final layer represents the Cloud, equipped with greater storage and processing capabilities. It functions as the central infrastructure for nationwide healthcare data management and coordination.

Representing patient data using RDF offers a structured and standardized method for managing healthcare information. RDF facilitates the integration of diverse data sources—including vital signs, medical histories, and IoT device readings—into a machine-readable format that promotes interoperability and semantic understanding. By organizing data into triples (subject, predicate, object), RDF captures the relationships between data points, enabling efficient querying, sharing, and analysis of patient information. This model supports effective decision-making, real-time monitoring, and system scalability, making it well-suited for modern healthcare environments.

The creation of RDF files marks the final step within the semantic layer, transforming raw IoT data into structured, meaningful, and actionable formats. After collecting health-related data, the system applies the SAREF ontology to standardize and organize information into a consistent, machine-readable structure. Furthermore, ontology-based SWRL rules are used to infer additional insights about the patient, aiding in the generation of relevant services and personalized recommendations. An inference engine is also incorporated to validate and enhance the accuracy of the generated knowledge. This seamless integration of semantic technologies improves both decision-making and service delivery by enabling more informed and context-aware actions. An example RDF file is shown in Figure 4, illustrating how patient data is organized and semantically enriched for practical use.

To support the early detection of health deterioration, the framework incorporates an Early Deterioration Identification System based on the Modified Early Warning Score (MEWS).



```
@prefix saref4health: <https://saref.etsi.org/saref4health/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix patient: <http://example.org/patient/> .
@prefix vital: <http://example.org/vitalSigns/> .

# Patient Information
patient:12345 a saref4health:Patient ;
    saref4health:hasName "Sami" ;
    saref4health:hasAge "45"^^xsd:integer ;
    saref4health:hasGender "Male" ;
    saref4health:hasMedicalRecordID "MR12345" .
# Vital Signs Measurements
vital:HeartRate_001 a saref4health:HeartRateMeasurement ;
    saref4health:hasValue "72"^^xsd:integer ;
    saref4health:hasUnit "bpm" ;
    saref4health:hasTimestamp "2023-10-15T08:30:00Z"^^xsd:dateTime ;
    saref4health:measuredBy patient:12345 .
vital:BloodPressure_001 a saref4health:BloodPressureMeasurement ;
    saref4health:hasSystolicValue "120"^^xsd:integer ;
    saref4health:hasDiastolicValue "80"^^xsd:integer ;
    saref4health:hasUnit "mmHg" ;
    saref4health:hasTimestamp "2023-10-15T08:30:00Z"^^xsd:dateTime ;
    saref4health:measuredBy patient:12345 .
# Device Information
vital:WearableDevice_001 a saref4health:WearableDevice ;
    saref4health:hasDeviceID "WD12345" ;
    saref4health:hasManufacturer "HealthTech Inc." ;
    saref4health:hasModel "SmartWatch X200" ;
    saref4health:monitors patient:12345 .
```

Figure 4: Patient's RDF file.

MEWS is a clinical scoring method that evaluates five vital signs: *blood pressure, pulse, oxygen saturation, respiratory rate, and body temperatur*e. Each vital sign is assigned a score from 0 to 3, where 0 indicates a normal state, and higher scores reflect increasing risk levels. Clinicians calculate a patient's overall MEWS score by summing the individual scores, resulting in a total value ranging from 0 to 14. Each incremental increase in the MEWS score signifies a greater health risk and may prompt a change in the treatment plan. For example, an increase of one point suggests heightened clinical concern requiring immediate medical attention. Additionally, RDF files are structured to categorize patient health status into predefined MEWS score ranges—such as [0–1], [1–2], ..., [13–14]—ensuring precise classification of patient conditions for continuous monitoring and timely intervention.

To apply our framework to the healthcare scenario, we establish a direct connection between the raw vital sign data collected via IoT devices and the fuzzy classification process. The MEWS system serves as the foundational structure. As previously described, MEWS assigns scores (ranging from 0 to 3) to specific intervals of vital signs (e.g., a pulse rate below 40 receives a score of 2, a value between 51–90 receives 0, and values above 130 receive a score of 3). These MEWS ranges and their associated scores form the direct basis for defining the inputs to our fuzzy logic system. In particular, the fuzzification process translates raw numerical vital sign measurements into fuzzy linguistic terms based on the MEWS-defined scoring ranges. For each vital sign—such as systolic blood pressure, heart rate, oxygen saturation, and temperature (i.e., the Attributes extracted from the RDF file)—we define a corresponding set of fuzzy linguistic terms aligned with MEWS scoring levels. For example, for heart rate, the terms might include: Low2 (corresponding to MEWS score 2 range <40), Low1 (score 1 range 41-50), Normal (score 0 range 51-90), High1 (score 1 range 91-110), High2 (score 2 range 111-130), and High3 (score 3 range >130).

Membership functions are then defined for each linguistic term, mapping the numerical value of a vital sign to a degree of membership (ranging from 0 to 1). As illustrated in Figure 5 (for pulse rate), these membership functions—trapezoidal—are shaped directly based on MEWS ranges.

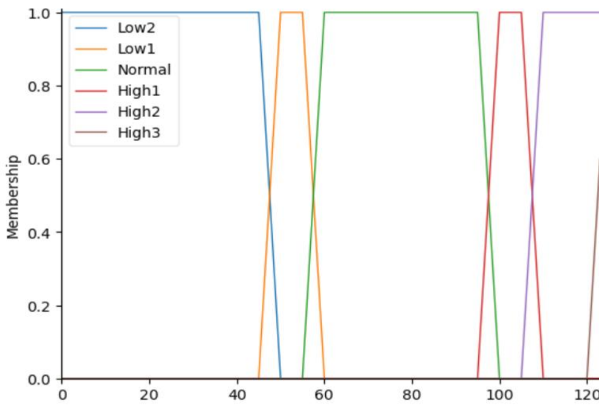


Figure 5 : Membership function of pulse rate

This allows for smooth transitions between categories rather than abrupt thresholds. For instance, a heart rate of 95 bpm may have a high degree of membership in High1, but also a small degree in Normal, reflecting uncertainty near clinical boundaries. This fuzzified representation of all relevant vital signs becomes the input to the fuzzy inference engine. The fuzzy inference engine applies a set of predefined fuzzy rules designed to mirror the clinical logic behind the MEWS system, but operating on fuzzy rather than crisp inputs. These rules—often implemented using Mamdani-style inference [40]—combine fuzzy states across multiple vital signs. For example: *IF Pulse is High1 AND OxygenSaturation is Low1 then Risk IS Moderate*. The results of these rules are aggregated into a final fuzzy output set that characterizes the patient's overall risk profile.

The final step, defuzzification—typically performed using the centroid method (see Equation 1)—converts the fuzzy output set into a single crisp value: the fuzzy MEWS score $S$. This score represents the system's comprehensive assessment of the patient's condition, analogous to the standard MEWS score but enhanced through fuzzy logic to better manage uncertainty and borderline cases. The calculated score is used to classify the patient into one of 15 predefined risk categories (see Table 3). For instance, a score between 7.0 and 8.0 may correspond to the *Elevated Risk* category. This process illustrates how raw healthcare data (vital signs) are transformed into fuzzy inputs, processed using clinically grounded rules, and translated into a decision-making score. The fuzzy score not only determines the patient's risk category $C_l^i$, but also guides the insertion of the patient's feature vector (comprising $S$ and original measurements) into the appropriate binary tree using Algorithm 2, directly integrating fuzzy classification into the retrieval framework. The shapes and boundaries of the membership functions were derived directly from the clinically validated MEWS scoring ranges. For example, the *Normal* pulse range ([51–90]) forms the core of the corresponding membership function, while trapezoidal shapes are used for transitional zones. These slightly overlap adjacent ranges (e.g., *Low1*, *High1*) to accurately model the ambiguity near clinical thresholds. This design ensures that the fuzzy representation remains grounded in established medical practice.

As shown in Figure 5, membership functions for vital signs are used as inputs to a set of predefined fuzzy rules structured in the form: *IF antecedent (input condition) THEN consequent (risk level).* These rules are based on combinations of vital signs such as heart rate or oxygen saturation, each represented by a linguistic term (e.g., *High*, *Low*) with associated membership degrees.

The fuzzy inference system uses Mamdani-style rules [40] and logical operators such as AND and OR to produce a fuzzy output set, which is then defuzzified into a single global MEWS score. This score corresponds to one of 15 predefined health statuses—ranging from *Optimal Health* to *Unknown Risk* —as outlined in Table 3. If a score does not fall within a category's defined range, the system proceeds to evaluate the next category until the appropriate classification is identified.

Introducing fuzzy logic into clinical systems requires careful design to ensure reliability, interpretability, and clinical alignment. Our approach incorporates four key safeguards:

1. Clinical Grounding: Membership functions and fuzzy rules were derived directly from MEWS guidelines, ensuring precise alignment with medical standards and avoiding the introduction of untested diagnostic logic.
2. Modeling Natural Uncertainty: Rather than introducing ambiguity, fuzzy logic models inherent uncertainty and gradual physiological transitions—features that crisp threshold fail to capture. This reduces misclassification risks for borderline cases.
3. Interpretability: Unlike black-box machine learning models, the rule-based structure of the fuzzy system allows clinicians to inspect, understand, and adjust the system logic. The use of clear linguistic terms and IF-THEN rules ensures transparency.
4. Expert Validation: A practicing physician can review selected patient cases to assess the accuracy and clinical relevance of the fuzzy logic-enhanced MEWS system. Such expert evaluations can confirm that the system often produces more nuanced and context-appropriate classifications—particularly in borderline scenarios where traditional MEWS scores may be ambiguous. This type of validation is essential to ensure that the fuzzy logic approach enhances, rather than compromises, the reliability of clinical risk assessment.

By grounding the fuzzy logic system in clinical knowledge, accurately modeling uncertainty, ensuring transparency, and validating outputs with expert feedback, our approach may deliver a robust and trustworthy enhancement to traditional MEWS scoring. It enables precise, interpretable, and clinically meaningful classification of patient risk in real-time healthcare environments.

It is important to clarify that the objective of incorporating fuzzy logic in our framework is not to perform unsupervised clustering (as with algorithms like k-means or DBSCAN), which aim to discover previously unknown data structures. Instead, we leverage fuzzy logic to enhance classification accuracy within the predefined, clinically meaningful MEWS risk categories. Physiological data inherently exhibits vagueness and gradual transitions between health states—characteristics that fuzzy logic is uniquely suited to model through the use of membership functions. In contrast, the rigid thresholds of standard MEWS scoring and the crisp assignments of traditional clustering algorithms are ill-equipped to represent these nuanced transitions. Moreover, standard clustering approaches do not utilize the clinical relevance embedded in the MEWS categories and are thus less suitable for this application. As a result, our primary evaluation focuses on demonstrating the improvement in classification performance achieved by integrating fuzzy logic, using the standard MEWS system as the direct clinical baseline for comparison.

Table 3: Health status corresponding to the defined ranges**.**

| Range | Category | Description |
|---|---|---|
| **[0–1]** | Optimal Health | All indicators are within excellent ranges. |
| **[1–2]** | Healthy Normal | Slight variations within the healthy range, no concerns. |
| **[2–3]** | Mild Variation | Slight deviation from the ideal range but still acceptable. |
| **[3–4]** | Low Alert | A single parameter slightly outside the normal range, minimal risk. |
| **[4–5]** | Moderate Alert | Two or more parameters are borderline abnormal, requiring monitoring. |
| **[5–6]** | Potential Concern | Mild irregularities across several parameters; action recommended. |
| **[6–7]** | Early Warning | Detectable issues that may indicate emerging problems. |
| **[7–8]** | Elevated Risk | Significant deviation in one or two key indicators; requires action. |
| **[8–9]** | Moderate Risk | Persistent deviations suggesting underlying conditions. |
| **[9–10]** | High Risk | Clear signs of health risk; needs immediate attention. |
| **[10–11]** | Critical Risk | Severe abnormalities; potential medical emergency. |
| **[11–12]** | Unstable | Multiple parameters indicate serious health instability. |
| **[12–13]** | Acute Danger | Immediate medical intervention required. |
| **[13–14]** | Severe Emergency | Life-threatening abnormalities across vital signs. |
| **[14–15]** | Unknown Risk | Insufficient or inconsistent data to assess risk level. |

In healthcare systems, efficient indexing of medical records is also essential, as it allows for rapid and accurate retrieval of patient data—crucial for timely decision-making and improved clinical outcomes. Effective indexing ensures that key information—such as health status, treatment history, and medical records—remains readily accessible, even within the vast and complex structure of healthcare datasets. Our indexing approach relies on constructing a Binary Tree, in which each node holds two pivot points. Given that RDF files are classified into 15 distinct categories, the system creates a separate Binary Tree for each category, resulting in a total of 15 trees. Algorithm 2 outlines the procedure for inserting a new RDF file into the corresponding Binary Tree. The input vector for indexing is constructed using the patient's vital sign measurements along with the fuzzy score generated during classification.

In the following section, we present a practical implementation using a patient RDF file containing vital sign information. This example demonstrates how an RDF file is inserted into and retrieved from a Binary Tree structure comprising 15 distinct categories. Each class is represented by its own Binary Tree, with each node maintaining two pivot vectors to support efficient similarity-based indexing.

1. Input: The patient RDF file (Figure 4)
2. Fuzzification

Fuzzification converts crisp input values into fuzzy sets using membership functions. Each vital sign ($v_i$) is mapped to a fuzzy linguistic term (e.g., Very Low, Low, Normal) based on its range.

$$\mu_A(x): X \to [0, 1]$$

Where:

- $\mu_A(x)$: Membership function of fuzzy set $A$ for input $x$
- $X$: The domain of input values (e.g., Pulse, Blood Pressure)

Example for Pulse:

$$\mu_{\text{VeryLow}}(x) = \begin{cases} 1, & x \le 40 \\ \frac{50-x}{10}, & 40 < x \le 50 \\ 0, & x > 50 \end{cases} \quad (5)$$

Combined Membership Function: Each vital sign's fuzzy score is calculated based on predefined ranges, producing membership values ($\mu$) for each linguistic category.

Fuzzy Score for Pulse=

$$\mu_{\text{VeryLow}}(x)+\mu_{\text{Low}}(x)+\mu_{\text{Normal}}(x)+\mu_{\text{High}}(x)+\mu_{\text{VeryHigh}}(x) \quad (6)$$

The fuzzy rule base consists of rules combining all vital signs:

Rule: If $\mu_{v1}$ is A and $\mu_{v2}$ is B and …, then $\mu_{\text{Output}} = C$

Where:

- $v_1,v_2,\ldots,v_n$ : Vital signs (e.g., Pulse, Temperature)
- $A,B,\ldots$: Membership values of each input
- $C$: Output class (e.g., Low Alert)

Example Rule:

If $\mu_{\text{Pulse}}$ is Normal and $\mu_{\text{Temperature}}$ is High, then Output = Moderate Alert

The fuzzy rules are then aggregated using fuzzy logic operators (e.g., AND, OR) to calculate the overall membership of the output.

Using AND (Minimum Operator):

$$\mu_{\text{Output}}=\min(\mu_{\text{Pulse}}, \mu_{\text{Temperature}},\ldots)$$

Using OR (Maximum Operator):

$$\mu_{\text{Output}}=\max(\mu_{\text{Pulse}}, \mu_{\text{Temperature}},\ldots)$$

3. Defuzzification

Defuzzification converts the fuzzy output into a crisp score ($S$), which represents the patient's overall status. The centroid method is commonly used:

$$S = \frac{\int_{\text{Range}} x \cdot \mu_{\text{Output}}(x)dx}{\int_{\text{Range}} \mu_{\text{Output}}(x)dx} \quad (7)$$

Where:

- $x$: Crisp value within the fuzzy range
- $\mu_{\text{Output}}(x)$: Membership function of the output

4. Euclidean Distance for Indexation

The fuzzy score ($S$) and vital sign values are combined into a vector (v) for comparison within the binary tree.

$\boldsymbol{v} = [S, v_1,v_2,\ldots,v_n]$

The Euclidean distance between two vectors ($\mathbf{v}_i$ and $\mathbf{v}_j$) is calculated as:

$$d(\mathbf{v}_i, \mathbf{v}_j) = \sqrt{\sum_{k=1}^{n}(\mathrm{v}_{i,k} - \mathrm{v}_{j,k})^2}$$

5. Binary Tree Search

Using the calculated distance, the binary tree is traversed to find the closest match:

1. Compare $\boldsymbol{v}$ to pivot nodes:

$$d_1 = d(\boldsymbol{v}, \text{Pivot1})$$
$$d_2 = d(\boldsymbol{v}, \text{Pivot2})$$

Move to the left or right child based on the smallest distance: If $d_1<d_2$, traverse left. Otherwise, traverse right.

A. Input Data

1. Patient Data (RDF File):
   - Measurement values: Pulse: 85; Blood Pressure: 120/80; Respiration Rate: 16; Body Temperature: 37°C
   - Fuzzy Score: 2.3 (calculated using fuzzy logic)
2. Binary Tree Example (Class "Healthy Normal"):
   - Root Node: Pivot 1: [85, 120, 16, 37, 2.0] and Pivot 2: [90, 125, 18, 38, 2.5]
   - Left Node: Pivot 1: [80, 115, 14, 36.5, 1.5] and Pivot 2: [82, 118, 15, 36.8, 1.8]
   - Right Node: Pivot 1: [92, 130, 19, 38.5, 3.0] and Pivot 2: [95, 135, 20, 39, 3.5]

B. Insertion Process

1. Determine the Fuzzy Score:
   - The fuzzy logic system calculates a score of 2.3, placing the RDF file in the class "Healthy Normal"
   - The corresponding Binary tree for "Healthy Normal" is used.
2. Construct the Input Vector:
   - Vector = [85, 120, 16, 37, 2.3].
3. Insert into Binary Tree (Algorithm 2):
   - Compare the vector to the Root Node pivots using Euclidean distance:
     - Distance to Pivot 1 = 0.3
     - Distance to Pivot 2 = 0.2

- The closest pivot is Pivot 2, so move to the Right Node.
- Compare the vector to the Right Node pivots:
  - Distance to Pivot 1 = 0.7
  - Distance to Pivot 2 = 1.0
- The closest pivot is Pivot 1, so the RDF file is inserted in the Right Node under Pivot 1.

C. Searching Process

1. Calculate the Fuzzy Score:
   - For retrieval, the fuzzy score is recalculated as 2.3 to determine the class "Healthy Normal"
2. Traverse the Tree (Algorithm 3):
   - Start at the Root Node and compare the vector to the pivots:
     - Distance to Pivot 1 = 0.3
     - Distance to Pivot 2 = 0.2
   - Move to the Right Node (closest pivot).
   - Compare the vector to the Right Node pivots:
     - Distance to Pivot 1 = 0.0 (match found).
   - The RDF file is retrieved successfully.

D. Outcome

1. Insertion Result:
   - As presented in Figure 6: Efficient and Scalable insertion of RDF file., the RDF file with vector [85, 120, 16, 37, 2.3] is inserted into the "Healthy Normal" Binary tree at the Right Node under Pivot 1.
2. Search Result:
   - The algorithm retrieves the RDF file from the Right Node based on the closest match to the input vector.

The algorithm demonstrates efficiency by narrowing down searches to specific pivots and clusters, reducing unnecessary computations. It is highly scalable, as new RDF files can be seamlessly added to the appropriate binary tree, ensuring that future searches maintain their efficiency even as data volumes grow. Additionally, the use of Euclidean distance calculations enhances accuracy by reliably retrieving the closest match, preserving the system's precision and ensuring consistent results.

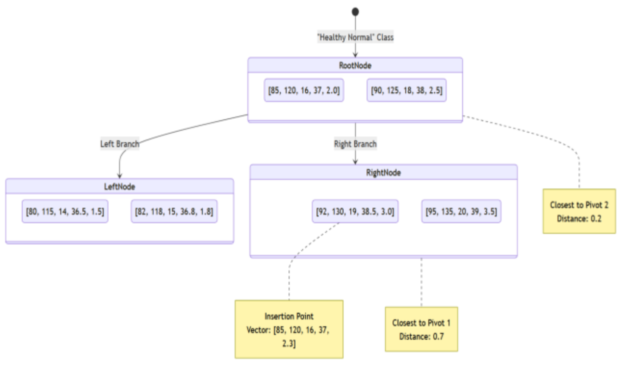


Figure 6: Efficient and Scalable insertion of RDF file.

## 6 Experimental evaluation

The integration of semantic technologies and fuzzy logic into IoT systems offers a powerful approach to enhancing data interoperability, accuracy, and efficiency. This study explores the implementation of a comprehensive framework designed to process, classify, and retrieve IoT data through the use of RDF files, semantic ontologies, and fuzzy logic. By leveraging these advanced computational techniques, the proposed framework effectively addresses key challenges such as data heterogeneity, imprecise sensor measurements, and the growing need for scalable storage and retrieval mechanisms. The results highlight the potential of combining semantic reasoning and fuzzy inference to manage the inherent complexity of IoT-driven systems—particularly in high-stakes environments like healthcare. Experimental validation confirms the robustness and efficiency of the proposed solution, emphasizing its practical applicability in real-world contexts. The outcomes pave the way for more reliable, intelligent, and semantically aware IoT architectures capable of supporting critical decision-making and delivering improved system performance at scale.

### 6.1 Data collection

To address the lack of publicly available datasets that accurately represent IoT-based patient health monitoring, the BIDMC dataset [41] —a subset of the larger MIMIC dataset [42] —was utilized. MIMIC is an open-access repository containing anonymized patient data collected from a U.S. hospital, offering a valuable resource for healthcare research and real-world evaluation. The dataset includes detailed records of patients' vital signs, such as blood pressure, oxygen saturation, heart rate, and other essential physiological metrics. The BIDMC data specifically consists of numerous files—often in formats such as CSV—which are derived from original waveform data [43]. For our experiments, we used a selection of these CSV files, each typically containing several hundred timestamped measurements for multiple patients. This structure served as the foundation for generating our experimental RDF datasets. In addition to vital sign data, the BIDMC dataset includes demographic information, such as patient age and gender, which enriches the dataset and supports a more comprehensive evaluation of patient health status. By leveraging the BIDMC dataset, our framework is grounded in realistic and diverse clinical data, ensuring a robust foundation for the development and testing of IoT-driven health monitoring solutions.

### 6.2 Methods

To evaluate the feasibility and performance of the proposed approach, the framework was implemented on a machine equipped with an Intel® Core™ i7-6700U CPU running at 3.40 GHz, with 8 GB of RAM and a 500 GB hard drive. The implementation was developed using Python, and the experiments were conducted on a Windows 8.1 operating system.

The transformation of selected BIDMC vital sign data (originally in CSV format) into RDF was carried out using custom Python scripts that utilized the Owlready[2] library and the SAREF ontology, specifically the SAREF4Health extension, to ensure semantic interoperability. The process included the following steps:

1. Measurement Extraction and RDF Modelling: Each source CSV file was parsed line by line. For every row—representing a timestamped set of vital sign measurements for a patient—a corresponding RDF structure was generated. This involved instantiating relevant SAREF4Health classes (e.g., saref4health:Patient, saref4health:Heart-Rate-Measurement, saref4health:Blood -Pressure -Measurement) and populating semantic properties such as saref4health:hasValue, saref4health:hasUnit, saref4health:hasTimestamp, while linking each measurement to its corresponding patient instance. The modelling approach closely followed the structure illustrated in Figure 4.
2. RDF File Generation: Each individual measurement instance—representing a specific vital sign recorded at a specific time for a specific patient—was saved as a separate, self-contained RDF file. This granular file generation strategy was adopted to simulate a realistic IoT scenario involving streams of sensor readings or discrete health events, while also testing the framework's ability to efficiently manage a large volume of distinct semantic data files.
3. Dataset Construction: Six experimental datasets of varying sizes were constructed based on the number of CSV files processed. Since each CSV file contained approximately 500 individual measurement instances, the datasets were generated in scalable increments to test the system under different load conditions:
   - Dataset 1 (~2,500 RDF files): Generated from processing ~5 source CSV files.
   - Dataset 2 (~5,000 RDF files): Generated from processing ~10 source CSV files.
   - Dataset 3 (~10,000 RDF files): Generated from processing ~20 source CSV files.
   - Dataset 4 (~15,000 RDF files): Generated from processing ~30 source CSV files.
   - Dataset 5 (~25,000 RDF files): Generated from processing ~50 source CSV files.
   - Dataset 6 (~50,000 RDF files): Generated from processing ~100 source CSV files.

This dataset generation method ensured that all experimental datasets shared a consistent RDF structure based on SAREF4Health and contained similar types of vital sign data. The primary controlled variable was the volume of RDF files, allowing a systematic evaluation of the framework's performance—particularly in terms of retrieval time and index scalability—under conditions that simulate the "Big IoT Data" challenge. These generated RDF files served as the direct input to the subsequent layers of the framework, with the *ExtractAttributes* step parsing the necessary vital sign values from each file for further processing. To rigorously evaluate the proposed framework, several performance metrics were employed. To enhance the accuracy of health status calculations, the `skfuzzy`[3] library was employed, leveraging a fuzzy logic algorithm. The evaluation focused on the proportion of cases reclassified into more appropriate risk categories, compared to the original standard MEWS classifications. This was quantified based on the percentage of patient records whose risk classification improved (as shown in Figure 5), supported by expert validation.

Traditional unsupervised clustering metrics were not used, as they are less suitable for this task driven by clinically grounded categories. Retrieval efficiency was primarily assessed through query execution time (in seconds). The performance of our binary tree indexing was compared against non-indexed searches and against established RDF management systems—namely Jena, RDFLib, Blazegraph, and DBgraph—across datasets of increasing size. While direct optimization of storage size was not a central focus, the scalability of the indexing structure was examined by tracking key characteristics of the binary trees—such as the number of internal nodes per level, tree height, and leaf node count—as dataset sizes increased. These structural metrics reflected the organizational efficiency that contributed to the framework's overall retrieval performance.

To benchmark our framework against traditional RDF data management systems, specific steps were taken to align the evaluation conditions, considering the architectural differences between our file-based retrieval model and the centralized triple-store architecture used by the baseline systems.

1. Data Consolidation: Unlike our system, which operates on discrete RDF files, traditional triple stores require data to be consolidated into a single RDF graph. Therefore, for each dataset size (e.g., 50,000 RDF files), we programmatically merged all individual RDF files into a single large RDF file using standard serialization formats (Turtle). This consolidation allowed the baseline systems to import and query the data using their native triple store engines. In contrast, our system bypasses this consolidation step, operating directly on individual files—a design that contributes significantly to its improved query performance.
2. Data Handling and Querying: Each baseline system imported the corresponding consolidated RDF file. Queries analogous to those used in our framework (e.g., retrieving patient records based on specific measurement values) were then formulated using SPARQL, the standard RDF query language. Query execution followed each system's native pipeline—SPARQL parsing, query plan optimization, index lookup (typically Subject-Predicate-Object), and result retrieval—which contrasts with the vector-based similarity search mechanism used in our system.

[2] https://bitbucket.org/jibalamy/owlready2

[3] https://pythonhosted.org/scikit-fuzzy/

3. System Configuration: All baseline systems were installed and executed on the same hardware used for testing our framework, ensuring fair comparison conditions. Tests were conducted using default configurations, without performance tuning. While additional tuning could improve results, default settings were chosen to provide a standardized benchmark representing typical out-of-the-box behavior.
4. Performance Metric: The primary comparison metric was average query execution time (in seconds), consistent with our framework's retrieval evaluation. This directly reflects the efficiency and responsiveness of the system under load, addressing one of the core challenges in large-scale IoT data environments.
5. Reporting Units: Our framework operates on individual RDF files, while traditional systems operate on RDF triples. For comparative purposes. This dual reporting bridges the gap between the two models: it illustrates the file-based scale of our system while providing triple count estimates to contextualize performance relative to traditional triple store literature. The evaluation demonstrates that our framework's efficiency stems not only from its use of fuzzy classification and indexing but also from its lightweight, decentralized file-processing model, which avoids the overhead of building and querying massive unified graphs.

Regarding baseline comparisons, our performance evaluation included runtime analyses against several established RDF-based database systems—Jena, RDFLib, Blazegraph, and DBGraph— demonstrating the efficiency of our framework's retrieval mechanism. With respect to alternative clustering algorithms such as k-means, DBSCAN, or hierarchical clustering, it is important to emphasize the distinct goal of our fuzzy logic component. As discussed earlier, we adopted the Modified Early Warning System (MEWS) as the foundation for patient classification, and enhanced it using fuzzy logic specifically to handle the uncertainty, variability, and gradual transitions inherent in physiological data—elements that conventional MEWS thresholds do not manage well. Our aim was not to discover latent clusters but to refine clinically relevant classifications.

Traditional clustering algorithms like k-means operate under the assumption of well-separated clusters and require a predefined number of clusters (k), which does not align with the progressive and overlapping nature of patient health states nor with the fixed MEWS categories. Similarly, DBSCAN, though effective in handling noise, is highly sensitive to its parameters (e.g., *epsilon* and *minPts*) and lacks the capacity to model the subtle, fuzzy boundaries that characterize adjacent clinical risk levels. Therefore, these algorithms are conceptually unsuitable for replacing the fuzzy logic module in our framework. They serve a different purpose—namely unsupervised cluster discovery—whereas our framework focuses on fuzzy classification refinement grounded in domain-specific clinical rules.

As such, a direct runtime comparison with these traditional clustering algorithms was not conducted, as they would not yield clinically meaningful outputs or support the semantic indexing and retrieval stages tailored to the MEWS classification scheme. Our evaluation instead focused on demonstrating the improvement fuzzy logic offers over the standard MEWS baseline and the efficiency gains of the complete framework against relevant RDF storage solutions.

## 6.3 Results and discussion

This section presents and discusses the experimental results evaluating the different components of the proposed framework, focusing on its core objectives: achieving scalable, efficient storage and retrieval of RDF data and improving classification accuracy through fuzzy logic within the healthcare application scenario.

### 6.3.1 Framework scalability and retrieval efficiency

A primary objective of this research was to address the scalability challenges associated with managing large volumes of RDF data generated by IoT systems. To this end, we evaluated the efficiency of our proposed binary tree indexing mechanism and compared the retrieval performance of the overall framework against standard methods and widely used RDF database systems.

Figure 7 and Figure 8 provide insights into the structural characteristics of the binary tree index as the dataset size increased, reaching up to 50,000 RDF files. Figure 7 illustrates the distribution of internal nodes per level, while Figure 8 tracks the growth in both the overall tree height and the number of leaf nodes.

The patterns observed in these figures indicate that the binary tree structure scales effectively with the growing number of RDF files. Importantly, the tree maintains a manageable height, which is critical for ensuring efficient search operations. These results affirm the organizational scalability of the indexing approach, supporting its suitability for large-scale, file-based semantic data retrieval in IoT environments.

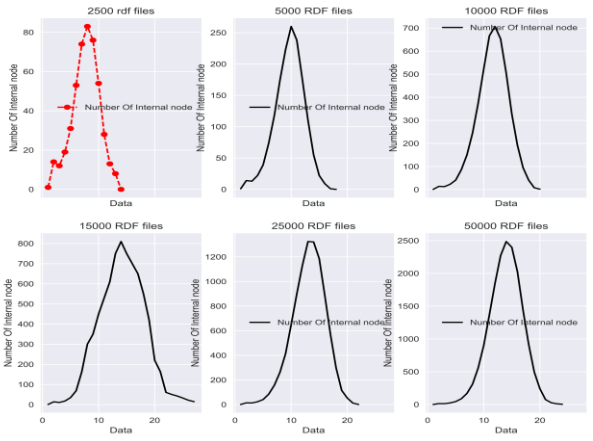


Figure 7: Number of internal nodes per level in the binary tree

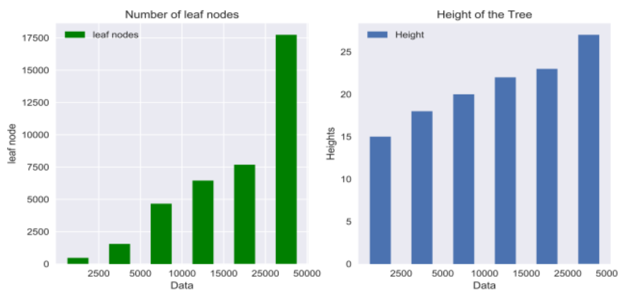


Figure 8: Comprehensive visualization of tree structure, height, and leaf node Count.

To quantify the efficiency gains achieved by the binary tree indexing method, we compared retrieval times against a baseline linear search approach conducted without the use of the index structure. This baseline simulated data retrieval by sequentially iterating through the relevant, classified RDF files—emulating a conventional non-indexed search. A set of 10 representative queries was executed for each dataset size, ranging from 2,500 to 50,000 RDF files. These queries were designed to retrieve specific RDF files based on randomly selected measurement values, with each query targeting feature vectors expected to be located at varied logical positions within the data structure (e.g., approximating beginning, middle, and end cases). This strategy was employed to capture potential variations in search performance based on data distribution. For each dataset size and query method (indexed vs. non-indexed), the search operation was repeated multiple times to ensure consistency. The average query execution time, measured in seconds, was then calculated and reported in Table 4.

Table 4: Comparative search time with and without binary tree indexing

| Size (Files) | 2500 | 5000 | 10000 | 15000 | 25000 | 50000 |
|---|---|---|---|---|---|---|
| **Without tree(s)** | 0.10 | 0.14 | 0.15 | 0.18 | 0.20 | 0.24 |
| **With tree (s)** | 0.003 | 0.004 | 0.005 | 0.0042 | 0.0055 | 0.0049 |

The results presented in Table 4 clearly demonstrate the substantial improvement in retrieval time achieved by utilizing the binary tree index, compared to the non-indexed baseline search. For the largest dataset comprising 50,000 RDF files, the average retrieval time dropped dramatically from 0.24 seconds to just 0.0049 seconds. The minor fluctuations observed in the "With Tree (s)" retrieval times are attributed to the dynamic nature of the binary tree's construction. Variability in data insertion order and the specific characteristics of the input vectors can affect tree balance and, consequently, the path length required to reach a given node during a search.

Despite this, the overall trend confirms the consistent and significant performance advantage provided by the indexing structure. Moreover, the initial fuzzy logic classification plays an important role in supporting this efficiency. By grouping RDF files into more homogeneous categories, it helps the indexing mechanism partition the search space more effectively, reducing the

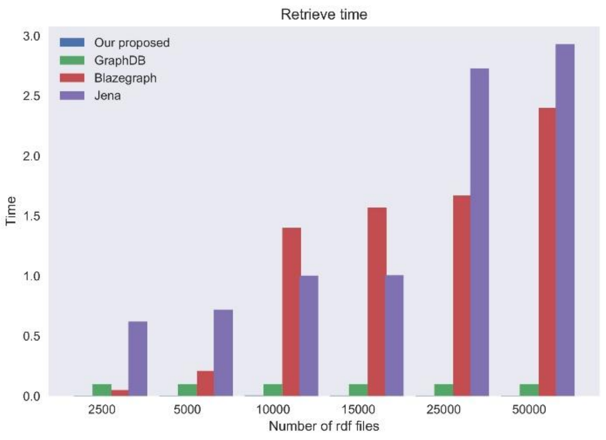


Figure 9: Comparison of the proposed framework with the RDF systems management.

complexity and time required for each query. This table offers a clear, numerical comparison that highlights the direct impact of the indexing component on retrieval performance, validating its role as a key contributor to the overall scalability and responsiveness of the proposed framework.

Figure 9 presents a critical comparison of the end-to-end query execution time of our approach against established RDF management systems—Jena, RDFLib, Blazegraph, and DBgraph—across datasets of increasing size. It is important to note that Figure 9 presents the overall retrieval process in each system, whereas Table 4 isolates the performance gain attributed specifically to the binary tree indexing component within our framework.

As detailed in Section 6.2, scalability was rigorously validated by extending tests up to 100,000 RDF files, representing several million RDF triples. The results clearly demonstrate that while traditional triple-store-based systems, which rely on complex SPARQL query processing over large, consolidated graphs, experience significant performance degradation at scale, our approach maintains consistently low and stable query execution times, even at 100,000 files. Notably, the performance at 100,000 files shows no appreciable increase compared to the 50,000-file dataset, underscoring the robustness of the architecture. This clear distinction—where baseline systems either struggled or became impractical due to SPARQL overhead and the limitations of triple management—strongly reinforces the superior scalability of our architecture, which combines fuzzy logic-based categorization with binary tree indexing.

### 6.3.2 Enhancement of classification accuracy via fuzzy logic

Within the healthcare application scenario, we evaluated the impact of integrating fuzzy logic into the MEWS classification process to better manage the inherent ambiguity present in vital sign data.

Table 5 presents selected examples comparing patient risk classifications generated using the standard MEWS system versus our fuzzy logic-enhanced approach. (BP = Blood Pressure, HR = Heart Rate, RR = Respiration Rate, OS = Oxygen Saturation, BT = Body Temperature). As illustrated, the fuzzy logic system yielded more nuanced scores in several cases, leading to different—and potentially more accurate—risk categorizations for patients (e.g., Patients 3, 4, and 5) compared to the rigid thresholds used in standard MEWS. These examples highlight the fuzzy system's ability to produce a more refined understanding of patient conditions, particularly in borderline scenarios.

Such differences underscore the risk of misclassification associated with traditional methods, which can be critical in clinical decision-making. By incorporating fuzzy logic, the classification process becomes more precise, offering clinicians deeper insights into patient status and supporting more informed interventions. Qualitative validation by an expert physician confirmed that the fuzzy-based classifications were, in many cases, more clinically representative than those derived from the standard MEWS system.

This expert feedback lends credibility to the practical utility of our approach in real-world healthcare contexts. Figure 10 further supports this observation by plotting the trend of 'Classes Improved' versus 'Classes Unchanged' as dataset size increases. The observed growth in 'Classes Improved' indicates that fuzzy logic consistently enhances classification outcomes across a substantial portion of cases. This improvement is particularly evident as data volume increases, demonstrating fuzzy logic's value in interpreting complex and imprecise patient data at scale. Ultimately, this improved classification accuracy directly contributes to the quality of data organization within our indexing structure, further enhancing the efficiency and clinical relevance of the overall retrieval process.

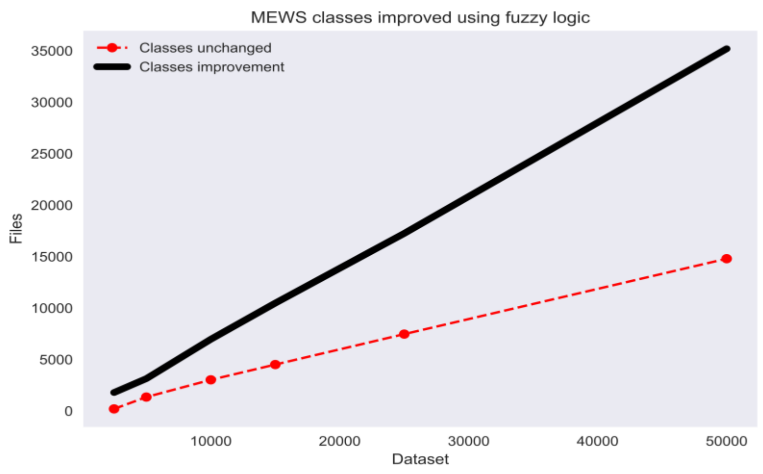


Figure 10: Comparison of MEWS classification with and without fuzzy logic.

Table 5: An example of classification of patients using the MEWS system with and without fuzzy logic

| Patient | Vital signs | | | | | | Results | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | Using MEWS | Using Fuzzy logic | |
| | BP | HR | RR | OS | BT | Score | Class | Score | Class |
| **1** | 229 | 70 | 35 | 74 | 38 | **7** | Moderate risk, further evaluation | **7.00** | Elevated Risk |
| **2** | 238 | 126 | 11 | 97 | 38 | **9** | Severe, urgent attention needed | **9.49** | High Risk |
| **3** | 201 | 105 | 19 | 67 | 38 | **8** | Moderate risk, further evaluation | **9.00** | High Risk |
| **4** | 215 | 116 | 20 | 81 | 38 | **6** | Early warning, intervention likely needed | **9.00** | High Risk |
| **5** | 182 | 118 | 19 | 90 | 38 | **7** | Moderate risk, further evaluation | **5.49** | Potential Concern |

### 6.3.3 Discussion on component roles and evaluation metrics

It is important to clarify the distinct roles of the framework's components in contributing to the observed performance improvements. The experimental results related to retrieval efficiency (presented in Table 4 and Figure 9) highlight the substantial speed gains primarily attributed to the combination of two mechanisms: fuzzy logic-based categorization (which narrows the initial search space) and subsequent binary tree indexing (which provides fast average access within the category). Together, these techniques directly address the challenge of efficiently navigating large-scale RDF datasets, making retrieval both scalable and responsive. While the semantic reasoning components—including SWRL rules and the inference engine described in Section 4—play a critical role in data enrichment. semantic consistency, and the provision of context-aware services (as illustrated in the healthcare scenario), are not directly involved in the core retrieval process measured in Algorithm 3. Therefore, the efficiency gains reported in this study stem primarily from the optimized data organization and indexing strategy, rather than from semantic reasoning modules.

Regarding evaluation metrics beyond execution time and classification comparison, we have not included standard metrics such as Precision, Recall, F1-score, and the Silhouette Score for specific methodological reasons. Calculating Precision /Recall /F1 rigorously requires a comprehensive ground truth dataset validated by multiple clinical experts, which was beyond the scope of this work. Our focus was on demonstrating the improvement over the standard MEWS baseline using fuzzy logic's ability to handle gradual transitions. Furthermore, Silhouette Scores, designed for unsupervised clustering, were deemed unsuitable. Our objective is not unsupervised cluster discovery but classification into predefined, clinically meaningful categories where adjacent risk levels are expected to overlap. Applying silhouette scores could yield misleadingly low values due to this inherent proximity, failing to reflect the clinical relevance and accuracy of the fuzzy classification.

This synergy between fuzzy classification and indexing directly contributes to the framework's adaptability in dynamic environments. Fuzzy logic robustly handles noisy or uncertain inputs and models gradual state transitions, providing a stable yet responsive classification. This accurate, adaptive categorization ensures that even evolving data is consistently organized within the appropriate index structure. Consequently, when decisions rely on retrieving relevant current or historical data, the optimized indexing allows for efficient access to information that accurately reflects the system's state, thereby enhancing the reliability and timeliness of the overall decision-making process.

The experimental results provide strong evidence supporting the effectiveness and scalability of the proposed multi-layered framework. The core research claims regarding scalability and efficient retrieval of large RDF datasets are directly addressed by the explicit numerical results in Table 4 (demonstrating the indexing benefit) and Figure 9 (showing superior performance compared to traditional RDF systems, validated up to 100,000 files). While evaluated within a healthcare use case, these performance results highlight the general applicability of the architecture. The fuzzy logic component demonstrably enhances classification accuracy within the chosen application by handling data ambiguity effectively. Together, these findings demonstrate that the framework offers a robust, scalable, efficient, and a promising approach to managing the complexities of SIoT data.

To mitigate potential data loss in dynamic, real-time IoT environments, the proposed architecture incorporates complementary standard practices, even though robust end-to-end loss prevention mechanisms fall outside the core algorithmic scope of this work. At the edge layer, buffering is employed to temporarily store generated RDF files during periods of network disruption, ensuring that data can be transmitted to the fog layer once connectivity is restored. This approach helps prevent immediate data loss at the source and supports continuity in data processing. In addition, both the fog and cloud layers inherently utilize reliable storage infrastructures that include redundancy and backup mechanisms, features commonly found in these tiers. These built-in safeguards protect the integrity of indexed and archived RDF files, even in the event of hardware or network failures. Although not explicitly detailed in the framework, a robust implementation would also incorporate transactional principles during data transfer and indexing to further ensure data consistency and reliability throughout the processing pipeline. Therefore, while the framework itself is primarily focused on efficient data processing and retrieval after reception, it implicitly relies on standard edge buffering and reliable storage strategies at the fog and cloud layers to uphold data integrity and minimize the risk of loss.

While cross-domain experimental testing represents crucial future work to definitively validate effectiveness elsewhere (like smart cities or industrial IoT, etc.), the framework's modular design and reliance on generalizable concepts (distributed architecture, semantic representation principles, fuzzy logic for uncertainty, vector indexing) provide a strong foundation for adaptability. The core data processing pipeline and indexing logic are expected to remain effective, with the primary adaptation effort concentrated on tailoring the semantic layer and the fuzzy classification system (rules, functions, etc.) to the specific requirements and knowledge of the target IoT domain.

## 7 Conclusion

This paper introduced and evaluated a comprehensive, multi-layered framework for the Semantic Internet of Things (SIoT), specifically designed to address key challenges in managing the heterogeneity, volume, and retrieval efficiency of Big IoT data represented in RDF format. Motivated by the need for greater scalability and improved accuracy compared to existing SIoT solutions and traditional RDF storage methods, we explored the integration of semantic representation, fuzzy logic classification, and binary tree indexing within a unified architectural model. The proposed framework leverages the SAREF ontology at the semantic layer to effectively standardize heterogeneous IoT data into RDF, thereby addressing core issues of data representation and interoperability. To manage and utilize the resulting RDF files efficiently, the framework introduces a novel combination of techniques within its fog layer. A central feature of the framework is the fuzzy logic component, which significantly enhances data classification accuracy. This improvement was particularly validated in a healthcare context, using the MEWS. Unlike traditional approaches based on crisp thresholds, the fuzzy logic mechanism is capable of handling ambiguity and gradual transitions, offering more nuanced and clinically relevant categorization of patient data. Following classification, the system produces optimized RDF file groupings, which are then processed by the indexing layer. Here, category-specific binary trees are employed to ensure efficient storage organization and rapid retrieval, overcoming the performance limitations typically associated with large-scale semantic datasets.

The framework’s effectiveness in achieving scalable and efficient data retrieval was validated through experimental evaluation. Performance comparisons demonstrated that the binary tree indexing significantly accelerates data retrieval compared to non-indexed searches, as shown by direct time measurements. Crucially, the complete framework exhibited substantially lower query execution times and superior scalability – maintaining stable performance up to 100,000 RDF files – when compared to established RDF management systems that rely on SPARQL over consolidated triple stores and show significant performance degradation at scale. These findings confirm that the proposed architecture offers significant improvements in efficiency and scalability for managing large-scale semantic IoT data. While the retrieval mechanism, based on vector similarity search, inherently avoids the need for user-formulated SPARQL queries, potentially enhancing usability for domain experts, a formal evaluation of this specific usability aspect was outside the scope of this study.

This work presents a viable and high-performance SIoT framework designed to effectively address the complexities associated with Big IoT RDF data. By integrating semantic standardization, fuzzy classification, and efficient binary tree indexing, the framework offers a validated, scalable, and efficient solution for data management. While the results are promising, we acknowledge that the current experimental validation is limited to a single domain. Further testing on diverse datasets from various IoT sectors is necessary to fully demonstrate the framework's broader applicability. Future work will focus on expanding the scope and adaptability of the approach. This includes exploring alternative clustering and indexing techniques, investigating the direct integration of semantic reasoning into the retrieval process, and conducting formal usability evaluations across a range of IoT domains. Additionally, a more comprehensive performance evaluation is warranted—one that extends beyond query execution time to include metrics such as data ingestion throughput, indexing time, and detailed measurements of computational overhead. Assessing these additional dimensions will provide a more complete characterization of the framework's efficiency and resource consumption, thereby further substantiating its practical advantages over baseline systems.

## References

[1] 'IoT devices installed base worldwide 2015-2025', Statista. Accessed: Apr. 25, 2025. [Online]. Available: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

[2] S. Benkhaled, M. Hemam, M. Djezzar, and M. Maimour, 'An Ontology – based Contextual Approach for Cross-domain Applications in Internet of Things', *Informatica*, vol. 46, no. 5, Mar. 2022, doi: 10.31449/inf.v46i5.3627.

[3] K. N. Prashanth Kumar, V. Ravi Kumar, and K. Raghuveer, 'A Survey on Semantic Web Technologies for the Internet of Things', in *2017 International Conference on Current Trends in Computer, Electrical, Electronics and Communication (CTCEEC)*, Mysore: IEEE, Sep. 2017, pp. 316–322. doi: 10.1109/CTCEEC.2017.8454974.

[4] J. D. McDonald and M. Levine-Clark, Eds., 'Resource Description Framework (RDF)', in *Encyclopedia of Library and Information Science, Fourth Edition*, 0 ed., CRC Press, 2017, pp. 3961–3969. doi: 10.1081/E-ELIS4-120043688.

[5] K. Gunaratna, S. Lalithsena, and A. Sheth, 'Alignment and dataset identification of linked data in Semantic Web', *WIREs Data Min. Knowl. Discov.*, vol. 4, no. 2, pp. 139–151, Mar. 2014, doi: 10.1002/widm.1121.

[6] M. H. Al-Zubaidie and R. H. Razzaq, 'Maintaining Security of Patient Data by Employing Private Blockchain and Fog Computing Technologies based on Internet of Medical Things', *Informatica*, vol. 48, no. 12, Sep. 2024, doi: 10.31449/inf.v48i12.6047.

[7] Y. Bu, 'Fuzzy Decision Support System for Financial Planning and Management', *Informatica*, vol. 48, no. 21, Nov. 2024, doi: 10.31449/inf.v48i21.6718.

[8] C. Hou, N. Xu, and S. Liu, 'Design of Online Monitoring Method for Distribution IoT Devices Based on DBSCAN Optimization Algorithm', *Informatica*, vol. 49, no. 5, Jan. 2025, doi: 10.31449/inf.v49i5.6399.

[9] X. Huo, 'Blockchain-Based Distributed Network Security Architecture with Smart Contract Vulnerability Detection Using Improved Tree CNN', *Informatica*, vol. 49, no. 17, Mar. 2025, doi: 10.31449/inf.v49i17.8050.

[10] F. Firouzi, B. Farahani, and A. Marinšek, 'The convergence and interplay of edge, fog, and cloud in the AI-driven Internet of Things (IoT)', *Inf. Syst.*, vol. 107, p. 101840, Jul. 2022, doi: 10.1016/j.is.2021.101840.

[11] H. A. Tran, D. Tran, L. G. Nguyen, Q. T. Ha, V. Tong, and A. Mellouk, 'SHIOT: A novel SDN-based framework for the heterogeneous Internet of Things', *Informatica*, vol. 42, no. 3, Sep. 2018, doi: 10.31449/inf.v42i3.2245.

[12] A. Rhayem, M. B. A. Mhiri, and F. Gargouri, 'Semantic Web Technologies for the Internet of Things: Systematic Literature Review', *Internet Things*, vol. 11, p. 100206, Sep. 2020, doi: 10.1016/j.iot.2020.100206.

[13] A. Gyrard, C. Bonnet, K. Boudaoud, and M. Serrano, 'LOV4IoT: A Second Life for Ontology-Based Domain Knowledge to Build Semantic Web of Things Applications', in *2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)*, Vienna, Austria: IEEE, Aug. 2016, pp. 254–261. doi: 10.1109/FiCloud.2016.44.

[14] M. Compton *et al.*, 'The SSN ontology of the W3C semantic sensor network incubator group', *J. Web Semant.*, vol. 17, pp. 25–32, Dec. 2012, doi: 10.1016/j.websem.2012.05.003.

[15] M. B. Alaya, S. Medjiah, T. Monteil, and K. Drira, 'Toward semantic interoperability in oneM2M

architecture', *IEEE Commun. Mag.*, vol. 53, no. 12, pp. 35–41, Dec. 2015, doi: 10.1109/MCOM.2015.7355582.

[16] L. Daniele, F. Den Hartog, and J. Roes, 'Created in Close Interaction with the Industry: The Smart Appliances REFerence (SAREF) Ontology', in *Formal Ontologies Meet Industry*, vol. 225, R. Cuel and R. Young, Eds., in Lecture Notes in Business Information Processing, vol. 225. , Cham: Springer International Publishing, 2015, pp. 100–112. doi: 10.1007/978-3-319-21545-7_9.

[17] 'Ontology (DBO)', DBpedia Association. Accessed: Nov. 28, 2024. [Online]. Available: https://www.dbpedia.org/resources/ontology/

[18] 'dblp /rdf'. Accessed: Nov. 28, 2024. [Online]. Available: https://dblp.org/rdf/

[19] 'Bio2RDF v2.7a'. Accessed: Apr. 25, 2025. [Online]. Available: https://bio2rdf.org/

[20] S. Duan, A. Kementsietsidis, K. Srinivas, and O. Udrea, 'Apples and oranges: a comparison of RDF benchmarks and real RDF datasets', in *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*, Athens Greece: ACM, Jun. 2011, pp. 145–156. doi: 10.1145/1989323.1989340.

[21] 'Lehigh University Benchmark (LUBM)'. Accessed: Apr. 25, 2025. [Online]. Available: https://swat.cse.lehigh.edu/projects/lubm/

[22] L. Ma, Z. Su, Y. Pan, L. Zhang, and T. Liu, 'RStar: an RDF storage and query system for enterprise resource management', in *Proceedings of the thirteenth ACM international conference on Information and knowledge management*, Washington D.C. USA: ACM, Nov. 2004, pp. 484–491. doi: 10.1145/1031171.1031264.

[23] D. J. Abadi, A. Marcus, S. R. Madden, and K. Hollenbach, 'SW-Store: a vertically partitioned DBMS for Semantic Web data management', *VLDB J.*, vol. 18, no. 2, pp. 385–406, Apr. 2009, doi: 10.1007/s00778-008-0125-y.

[24] 'Jena Property Table Implementation'. Accessed: Apr. 25, 2025. [Online]. Available: http://shiftleft.com/mirrors/www.hpl.hp.com/techreports/2006/HPL-2006-140.html

[25] 'Apache Jena - Home'. Accessed: Apr. 25, 2025. [Online]. Available: https://jena.apache.org/

[26] 'Blazegraph Database'. Accessed: Apr. 25, 2025. [Online]. Available: https://blazegraph.com/

[27] S. Sakr and A. Y. Zomaya, Eds., 'Graph Databases', in *Encyclopedia of Big Data Technologies*, Cham: Springer International Publishing, 2019, pp. 835–835. doi: 10.1007/978-3-319-77525-8_100147.

[28] S. Benedict, 'IoT-Enabled Remote Monitoring Techniques for Healthcare Applications -- An Overview', *Informatica*, vol. 46, no. 2, Jun. 2022, doi: 10.31449/inf.v46i2.3912.

[29] A. Cimmino *et al.*, 'VICINITY: IoT Semantic Interoperability Based on the Web of Things', in *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*, Santorini Island, Greece: IEEE, May 2019, pp. 241–247. doi: 10.1109/DCOSS.2019.00061.

[30] A. Broring *et al.*, 'The BIG IoT API - Semantically Enabling IoT Interoperability', *IEEE Pervasive Comput.*, vol. 17, no. 4, Art. no. 4, Oct. 2018, doi: 10.1109/MPRV.2018.2873566.

[31] A. Gyrard, C. Bonnet, K. Boudaoud, and M. Serrano, 'LOV4IoT: A Second Life for Ontology-Based Domain Knowledge to Build Semantic Web of Things Applications', in *2016 IEEE 4th International Conference on Future Internet of Things and Cloud (FiCloud)*, Vienna, Austria: IEEE, Aug. 2016, pp. 254–261. doi: 10.1109/FiCloud.2016.44.

[32] M. G. Kibria, S. Ali, M. A. Jarwar, and I. Chong, 'A framework to support data interoperability in web objects based IoT environments', in *2017 International Conference on Information and Communication Technology Convergence (ICTC)*, Jeju: IEEE, Oct. 2017, pp. 29–31. doi: 10.1109/ICTC.2017.8190935.

[33] D. Lymperis and C. Goumopoulos, 'SEDIA: A Platform for Semantically Enriched IoT Data Integration and Development of Smart City Applications', *Future Internet*, vol. 15, no. 8, Art. no. 8, Aug. 2023, doi: 10.3390/fi15080276.

[34] A. Pliatsios, D. Lymperis, and C. Goumopoulos, 'S2NetM: A Semantic Social Network of Things Middleware for Developing Smart and Collaborative IoT-Based Solutions', *Future Internet*, vol. 15, no. 6, Art. no. 6, Jun. 2023, doi: 10.3390/fi15060207.

[35] M. Banane, A. Belangour, and L. El Houssine, 'Storing RDF Data into Big Data NoSQL Databases', in *Lecture Notes in Real-Time Intelligent Systems*, vol. 756, J. Mizera-Pietraszko, P. Pichappan, and L. Mohamed, Eds., in Advances in Intelligent Systems and Computing, vol. 756. , Cham: Springer International Publishing, 2019, pp. 69–78. doi: 10.1007/978-3-319-91337-7_7.

[36] C. K. Wu *et al.*, 'An IoT Tree Health Indexing Method Using Heterogeneous Neural Network', *IEEE Access*, vol. 7, pp. 66176–66184, 2019, doi: 10.1109/ACCESS.2019.2918060.

[37] M. D. Le Lagadec, T. Dwyer, and M. Browne, 'Indicators of patient deterioration in poorly resourced private hospitals: Which vital sign to watch? A retrospective case–control study', *Aust. Crit. Care*, vol. 37, no. 3, pp. 461–467, May 2024, doi: 10.1016/j.aucc.2023.05.006.

[38] S. Nasiri, F. Sadoughi, A. Dehnad, M. H. Tadayon, and H. Ahmadi, 'Layered Architecture for Internet of Things-based Healthcare System: A Systematic Literature Review', *Informatica*, vol. 45, no. 4, Dec. 2021, doi: 10.31449/inf.v45i4.3601.

[39] M. Belkebir, T. M. Maarouk, and B. Nini, 'Realtime Semantic Healthcare System: Visual Risks Identification for Elders and Children', *Informatica*, vol. 48, no. 14, Sep. 2024, doi: 10.31449/inf.v48i14.6271.

[40] J. Martinez-Gil and J. M. Chaves-Gonzalez, ‘Interpretable ontology meta-matching in the biomedical domain using Mamdani fuzzy inference’, *Expert Syst. Appl.*, vol. 188, p. 116025, Feb. 2022, doi: 10.1016/j.eswa.2021.116025.

[41] M. Pimentel *et al.*, ‘BIDMC PPG and Respiration Dataset’. physionet.org, 2018. doi: 10.13026/C2208R.

[42] A. L. Goldberger *et al.*, ‘PhysioBank, PhysioToolkit, and PhysioNet: Components of a New Research Resource for Complex Physiologic Signals’, *Circulation*, vol. 101, no. 23, Jun. 2000, doi: 10.1161/01.CIR.101.23.e215.

[43] ‘Respiratory Rate Estimation by peterhcharlton’. Accessed: Apr. 25, 2025. [Online]. Available: https://peterhcharlton.github.io/RRest/datasets.htm l