Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Flexible Data Access Mechanisms for IoT Ecosystems: From Content-Based Addressing to LLM-Driven Data Acquisition

Codruta Maria Serban

Codruta.Serban@cs.utcluj.ro

Technical University of Cluj-Napoca

Gheorghe Sebestyen

Technical University of Cluj-Napoca

Anca Hangan

Technical University of Cluj-Napoca

Research Article

Keywords: IoT, flexible data access, content-based addressing, role-based access control, Large Language Models

Posted Date: November 19th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10586771/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Flexible Data Access Mechanisms for IoT Ecosystems: From Content-Based Addressing to LLM-Driven Data Acquisition

Codruta Maria Serban[1*], Gheorghe Sebestyen[1] and Anca Hangan[1]

[1]Department of Computer Science, Technical University of Cluj-Napoca, Memorandumului Street 28, Cluj-Napoca, 400114, Cluj, Romania.

*Corresponding author(s). E-mail(s): Codruta.Serban@cs.utcluj.ro;
Contributing authors: Gheorghe.Sebestyen@cs.utcluj.ro;
Anca.Hangan@cs.utcluj.ro;

Abstract

The continuous development and integration of public and private Internet of Things (IoT) infrastructures require flexible and scalable data access mechanisms that ensure seamless access to IoT data regardless of its source of physical address. Traditionally, retrieving data from different physical sources requires rigid, topology-dependent protocols and high technical skills; this may limit accessibility for a wider range of end-users.

Content-Based Addressing (CBA) accesses data based on its attributes or content rather than its network location. Users need an intuitive mechanism to extract IoT values from multiple sources (e.g. public and private sources), removing the need for technical database query knowledge. In a similar way to web search (e.g. Google), users ask questions in natural language based on content, location and time without direct reference to the physical location of the IoT devices that provide the necessary information.

This paper proposes a novel, hybrid framework for flexible IoT data access that combines CBA with a Large Language Model (LLM)-based conversational interface. The architecture is built on two core pillars: a Content-Addressable Storage (CAS) layer, coupled with a device registry, that unifies heterogeneous data sources, public APIs and private edge devices, into a single content-addressed store secured by a granular Role-Based Access Control (RBAC) mechanism. The second pillar is a Natural Language Query Engine that translates chat-based interrogations into executable system queries.

An adaptive imputation service maintains data completeness by detecting missing sensor readings, predicting replacement values, and reconciling predictions

when delayed measurements arrive. Together, the framework enables an intuitive experience for IoT data exploration with no technical training needed.

Keywords: IoT, flexible data access, content-based addressing, role-based access control, Large Language Models

1 Introduction

The Internet of Things (IoT) has experienced rapid growth over the past decade, connecting over 75 billion devices by 2025 [10], generating geographically distributed data streams from both private edge deployments and publicly accessible platforms. However, data retrieval in conventional IoT architectures remains tightly coupled to the physical topology of the network: accessing sensor readings requires knowledge of device identifiers, network endpoints, communication protocols, and query languages. This topology-dependent paradigm imposes significant technical barriers, restricting IoT data access to users with specialized engineering knowledge and limiting broader adoption of IoT analytics. The challenge is compounded in multi-source environments, where private edge devices coexist alongside public data providers, each exposing data through different protocols, schemas, and access policies. Integrating these sources into a unified, queryable data space while preserving data integrity, enforcing access control, and maintaining usability remains an open problem.

Content-Based Addressing (CBA) offers a first layer of flexibility by deriving each record's address from its content via a cryptographic hash, rather than from its network location. This decouples data identity from its physical source, providing automatic deduplication and integrity verification regardless of the ingestion channel. However, CBA alone does not address the usability gap: end-users rarely know the cryptographic key of the record they need.What they require is the ability to express queries in natural language: "What was the temperature in Cluj-Napoca last week?" or "Show me readings above a certain threshold from my organization's devices.". Recent advances in Large Language Models (LLMs), particularly their function-calling capabilities [17], enable a second layer of flexibility: a conversational interface that interprets natural-language questions, decomposes them into structured query parameters (metric type, geographic location, temporal range, value bounds), and invokes the corresponding system functions, without requiring knowledge of the underlying schema or query syntax. In multi-tenant environments, these mechanisms must be complemented by access control; Role-Based Access Control (RBAC) ensures that the flexibility achieved through CBA and natural-language querying does not compromise data governance.

This paper proposes a hybrid framework for flexible IoT data access that combines three complementary mechanisms: (i) a Content-Addressable Storage (CAS) layer that unifies heterogeneous data sources under a single, location-independent addressing scheme; (ii) a granular RBAC system that enforces organization-scoped, permission-based access control over all data operations; and (iii) an LLM-driven Natural Language (NL) Query Engine that translates conversational questions into

executable, multi-dimensional queries. An adaptive imputation service maintains data completeness across these layers by detecting missing readings and predicting replacement values using temporal and spatial models.

To validate the proposed framework, we implement and evaluate it in the domain of environmental monitoring. The prototype integrates private edge devices, ESP8266 microcontrollers equipped with DHT11 temperature/humidity sensors and MQ7 gas sensors, communicating over MQTT, alongside public air quality data from the OpenAQ platform ingested via its REST API. Data from both sources are stored in a unified content-addressable store, and users interact with the system through a conversational interface, querying across all sources using natural language.

The main contributions of this paper are as follows:

- A content-addressable IoT data platform that integrates private edge devices and public data sources into a unified store, where each record is immutably addressed by the SHA-256 hash of its canonical content, providing automatic deduplication and integrity verification.
- A role-based access control mechanism tightly integrated with content-addressable storage, supporting organization-scoped read permissions, per-user permission sets, and centralized authorization enforcement. It enables secure multi-tenant data sharing without exposing data across organizational boundaries.
- An adaptive imputation service that maintains data completeness in the content-addressable store by detecting missing hourly readings and predicting values using adaptive method selection (autoregressive, spatial regression, and spatial neural network), with cubic spline reconciliation when delayed real measurements arrive.
- A natural language query engine that leverages LLM function-calling to translate conversational questions into structured, multi-dimensional queries. The engine precomputes temporal reference tables to avoid LLM arithmetic errors and supports filtering by metric, device, organization, city, country, value range, time window, and geographic proximity.
- A prototype implementation and evaluation demonstrating the end-to-end operation of the proposed framework in an environmental monitoring scenario, from sensor data ingestion through encrypted MQTT and HTTP channels to natural language querying through a conversational interface.

The remainder of this paper is organized as follows. Section 2 reviews related work in content-based addressing, RBAC and LLM-based data interfaces. Section 3 presents the architecture and design of the proposed framework. Section 4 describes the details of implementation. Section 5 presents the evaluation and experimental results. Section 6 discusses the limitations and directions for future work, and Section 7 concludes the paper.

2 Related work

Implementing a flexible IoT data access framework requires abstracting the underlying network complexity to facilitate intuitive interrogation. Current literature addresses

this through two primary technical perspectives: structural decoupling (removes physical address dependencies) and semantic decoupling (bridges the gap between technical protocols and human needs).

The primary barrier to flexible IoT data access is the rigid coupling of data retrieval to specific network topologies. Traditional IoT architectures rely on IP-centric or URI-centric paradigms that tie information to a physical or logical address. This dependency creates fragile data pipelines; when devices migrate, disconnect, or are replaced, location-dependent systems suffer from broken references and high reconfiguration overhead [9]. Structural decoupling has emerged as a critical solution to this problem, functioning by abstracting the "where" of data storage from the "what" of the data content. To address this, Information-Centric Networking (ICN) and Content-Based Addressing (CBA) have emerged as primary enablers for structural decoupling.

Content-based addressing involves identifying and accessing data based on its content rather than its location or physical address, which may not be known. It is particularly useful in distributed systems like IoT, where data may be scattered across multiple devices or nodes. This allows for more efficient and flexible data retrieval.

Content-based addressing in IoT systems can be implemented at multiple layers of the solution. Information-Centric Networking (ICN) [20] and Named Data Networking (NDN) [16] architectures, retrieve content by name rather than source address, with in-network caching as a key feature. They're implemented at the network routing and caching layer. Another implementation of CBA that does not require special network infrastructure is CAS, located at the storage layer.

CAS in IoT systems uses cryptographic hash functions to obtain a unique, deterministic address for each data record from the record's content itself. Both the address and the data record (serialized as canonical JSON with sorted keys and minimal whitespace) are stored in the system's database. StorSec [21] describes IPFS, a distributed storage system for IoT that returns hash-ID for downloading and file-sharing purposes, enabling content-based retrieval. [7] proposes a hashing-based content searching approach combined with NDN to reduce content lookup time in IoT networks. The solution achieved a peak package rate of approximately 960 packets per second. For encryption, [19] demonstrates that "hash storage encryption of data provides a robust means for managing data securely within an IoT framework," specifically using SHA-256 algorithms.

Securing data access in multi-tenant IoT deployments requires an authorization model that balances overhead with the computational constraints of edge hardware. Role-Based Access Control (RBAC) remains the most widely deployed model due to its administrative simplicity, well-understood semantics and effectiveness [14]. RBAC alone may not properly address edge hardware constraints, therefore [8] and [11] propose attribute-based approaches for constrained devices. Attribute-Based Access Control (ABAC) is an authorization model where access decisions are based on attributes: properties attached to the user, the resource, the action, and the environment. [1] demonstrates that "the need arises for a hybrid model combining ABAC and RBAC," developing two hybrid approaches (HyBAC_RC and HyBAC_AC) to better

meet IoT requirements. . However, even with robust structural decoupling and fine-grained access control in place, users must still formulate queries in terms of device identifiers, field names, and API parameters, a barrier that limits data accessibility to technically skilled operators.

The integration of Large Language Models (LLMs) into IoT ecosystems addresses this remaining gap. It represents a fundamental shift in how users interact with complex data environments. Traditionally, retrieving insights from heterogeneous sensor networks required significant technical expertise in database interrogation, specialized APIs, or domain-specific protocols. By serving as a semantic bridge, LLMs enable Natural Language Interaction (NLI), allowing non-technical stakeholders to perform complex data exploration through intuitive conversational interfaces. This democratization of data access is particularly vital in time-critical or high-stress environments where the speed of insight is as important as the accuracy of the data itself.

The initial challenge in this domain was the transition from qualitative chat to quantitative data retrieval. SensorChat [18] addresses this by enabling LLMs to answer both qualitative and quantitative questions about sensor streams. However, while SensorChat successfully bridges the gap between raw data and human-readable answers, it operates primarily as an interrogation tool without a heavy focus on the underlying architectural security or the constraints of the hardware running the queries. To move these capabilities from the cloud to the field, researchers have explored the deployment of NLI on resource-constrained devices. In the context of the Internet of Battlefield Things (IoBT), [12] demonstrates the feasibility of hosting natural language database interfaces directly on edge devices. This work addresses a critical gap left by cloud-centric models, resilience in disconnected environments; but identifies a new bottleneck: the high computational cost of running localized LLMs for SQL translation.

In order to satisfy the need for efficiency and security, recent frameworks have focused on the Text-to-SQL translation pipeline. A secure and efficient LLM-based system proposed in [6] introduces specialized modules to translate natural language queries into SQL while maintaining a focus on IoT data management. This approach improves upon previous models by optimizing the translation speed, yet it highlights a persistent vulnerability: the potential for SQL injection or unauthorized data exposure if the LLM is not "wrapped" in a robust security layer.

The complexity of these interactions further increases when considering multimodal data and cross-protocol orchestration. Research in [3] explores the integration of LLMs with specialized sensor ontologies, allowing the model to act not just as a translator, but as an orchestrator that can reason across different types of sensor data. Finally, the work presented in [2] addresses the "Trustworthiness" gap by evaluating how LLMs can be utilized for anomaly detection and device identification within the network itself. This effectively closes the loop between access and integrity, suggesting that the LLM is most powerful when it is aware of the underlying network security state.

Together, these studies illustrate a clear evolution: from the qualitative capabilities of SensorChat [18], to the edge-resilience of IoBT systems [12], and finally to the secure, specialized translation and orchestration frameworks of modern IoT-LLM research [6] [3] [2] . It justifies the need for a unified architecture that combines semantic decoupling

with the structural decoupling and identity-aware security (RBAC) proposed in this paper.

3 Architecture and design

This section presents the architecture of the proposed flexible data access framework for IoT ecosystems. The design is driven by three guiding principles: (i) data should be addressed by its content rather than its network location; (ii) data retrieval should be accessible through natural language without requiring query language expertise; and (iii) access control should be enforced transparently, regardless of the interface used to query data.

3.1 System overview

The proposed framework is organized around six core components, illustrated in Fig.1:

1. CAS: the data persistence layer, which stores all IoT records under content-derived addresses, providing source-independent storage and automatic deduplication.
2. Device Registry: a metadata catalogue that associates each data-producing device with its geographic location, owning organization, and source type.
3. User and authorization service: a centralized authentication and permission enforcement point that governs all data operations.
4. Ingestion service: the entry point for heterogeneous data sources, responsible for normalizing and routing records into the CAS.
5. Imputation service: a periodic gap-filling component that detects missing sensor readings and predicts replacement values using adaptive methods, maintaining data completeness in the CAS.
6. Natural Language Query Engine: a conversational interface that translates user questions into structured, multi-dimensional queries.

A seventh component, the User interface, provides a web-based front-end with both a traditional dashboard view and a conversational query mode.

The architecture follows a principle of convergent ingestion and uniform retrieval: data enters the system through multiple, source-specific ingestion paths but, once stored, is accessed through a single set of content-based retrieval operations, subject to uniform access control. This convergence is the architectural foundation of flexible data access, the retrieval layer is entirely unaware of the data's origin, protocol, or ingestion path.

3.2 Content-Addressable Storage layer

The CAS layer addresses the first barrier to flexible data access: the coupling between data identity and data source. In conventional IoT architectures, records are identified by a combination of device identifier, endpoint address, and sequence number, identifiers that embed the data's provenance into its retrieval path. The CAS layer replaces this source-dependent addressing with content-derived addressing:

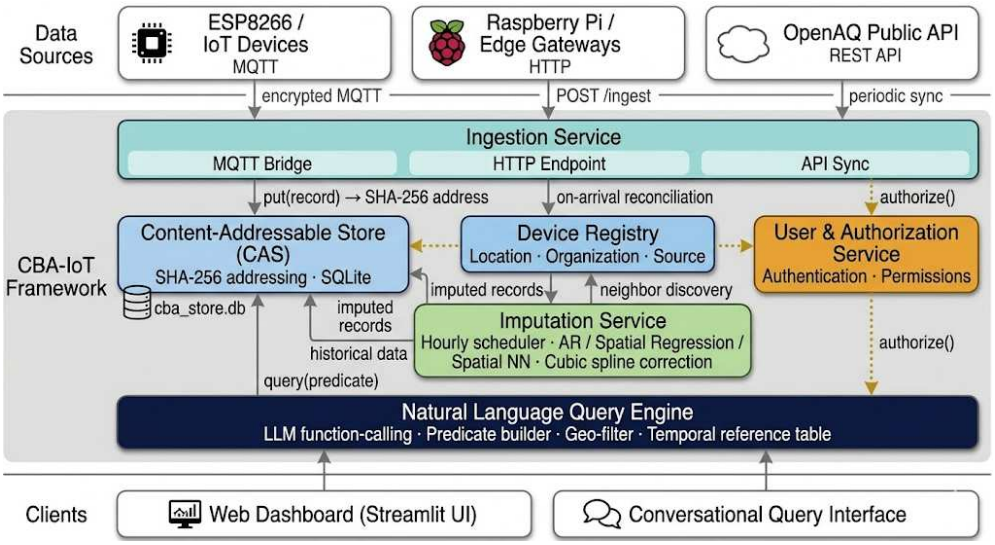


Fig. 1: Architecture diagram for the proposed framework

each record's unique identifier is computed deterministically from its content using a cryptographic hash function.

This design yields three properties relevant to flexible data access:

- Source independence: a temperature reading from a private sensor and a temperature reading from a public API are stored and retrieved using the same addressing scheme. The retrieval mechanism does not need to know how the data entered the system.
- Automatic deduplication: identical records, regardless of how many times produce the same content hash and are stored exactly once. This is particularly valuable in IoT environments where message retransmission and overlapping API fetches can produce duplicate data.
- Integrity verification: any stored record can be verified by recomputing its content hash and comparing it to the stored address, providing a built-in tamper-detection mechanism without requiring additional cryptographic infrastructure.

The CAS layer supports three retrieval modes, each offering a different level of flexibility. Firstly, it's the **exact lookup by content address**. It ensure direct retrieval when the record's hash is known, with constant-time performance. **Predicate-based query** consists in retrieval of all records whose content satisfies an arbitrary filter condition. This is the primary flexible access mechanism: queries can filter on any field or combination of fields present in the record without requiring predefined indices or schema modifications. A query such as "all records where metric is temperature, value exceeds 30, and device belongs to organization X" is expressed as a predicate evaluated against each record's content. The final retrieval mode is the **range query**. This is a specific type of predicate query designed for numeric fields, making it easier to perform common filtering tasks.

The predicate-based model represents a design trade-off: query latency scales linearly with the number of stored records, but the system can filter on any attribute without index maintenance. For edge IoT deployments where data volumes are huge and queries are user-initiated rather than high-frequency automated operations, this trade-off favors flexibility over throughput.

3.3 Device Registry

The Device Registry provides spatial context for content-based queries. While the CAS layer stores data independently of its source, many user queries are inherently geographic: "What is the temperature in Cluj-Napoca?". Answering these queries requires a mapping between device identifiers (which appear in records) and geographic locations, from user questions.

The registry associates each device with a unique identifier and owning organization, geographic coordinates (latitude, longitude) and human-readable location (city, country) and a source type distinguishing private IoT devices from public data sources. This metadata enables two forms of geographic query resolution:

- City/country resolution: when a user query mentions a location by name, the registry resolves it to the set of device identifiers deployed in that location.
- Proximity filtering: when a query specifies geographic coordinates and a search radius, the system computes the great-circle distance (Haversine formula) between the query center and each device's registered location, retaining only records from devices within the specified radius.

Both mechanisms decouple the user's spatial intent from the physical device topology: the user asks about a location, and the system determines which devices are relevant.

3.4 User Service and access control model

Flexible data access must also be secure data access. In multi-tenant IoT environments where multiple organizations share a common platform, unrestricted flexibility would compromise data isolation. The User Service provides the architectural foundation for secure access by serving as the single point of authentication, user management, and authorization enforcement in the framework.

The User Service is an independent component responsible for three functions: managing user accounts, credentials, and organizational memberships; authenticating users upon login and returning their identity context (username, role, organization, associated device, and permission set); and (iii) authorizing individual data operations by evaluating a user's permissions against the requested action.

By isolating these responsibilities in a dedicated service, rather than embedding them in the data store or the query engine, the architecture decouples functional roles having two benefits. First, the authorization policy can be updated, audited, or replaced independently of the data storage layer. Second, all access interfaces (the programmatic API, the NL query engine, and the web dashboard) enforce the same policy

by delegating to the same service, eliminating the risk of inconsistent enforcement across access paths.

Upon successful authentication, the User Service returns a user context that the client-side RBAC layer wraps into a session object. All subsequent data operations (reads, writes, deletions, device listing, user management) are invoked through this session. The session acts as a capability boundary: it holds the user's permission set and transparently intercepts every operation to request authorization from the User Service before proceeding. This design means that application code and the NL engine interact with a single session interface; the permission enforcement is invisible to the caller.

The framework implements a Role-Based Access Control model augmented with organization-scoped data isolation. Roles are convenience labels; what actually gates each operation is whether the user has the required permission [4]. Rather than relying solely on role labels, the system defines a set of fine-grained permissions (write, delete, read all data, read own organization's data, list devices, manage users) that can be combined into arbitrary permission sets per user. Default roles (administrator, standard user) are provided as convenience templates, but the enforcement mechanism operates on the permission set, not the role.

For read access, the authorization layer additionally constrains visibility based on the requesting user's organization membership. The *read_own* permission implements this scoping mechanism: queries executed by users with this permission are transparently filtered to return only records belonging to the user's organization, plus records from public sources. This scoping is enforced at the session level by wrapping the user's query predicate with an organization filter before execution. The user, and the NL engine acting on behalf of the user, are not aware of this filtering; it is applied uniformly regardless of the access interface. This organization check constitutes an attribute-based constraint, matching a resource attribute (organization) against a user attribute (membership). This characteristic is borrowed from ABAC, while the overall model retains the administrative simplicity and predictable enforcement cost of RBAC.

Users may belong to multiple organizations, each with a designated organizational role (e.g., viewer or editor) that further constrains write and delete operations within that organization. This supports scenarios where a user participates in several monitoring projects but should only have write access to a subset of them.

Every data operation triggers an authorization request to the User Service before proceeding. This per-operation enforcement model provides a single verification control point and ensures that permission changes, such as revoking a user's write access, take effect immediately, without client-side caching or propagation delays.

The combination of flexible content-based querying and transparent access control scoping means that two users issuing the same natural language question may receive different result sets, not because the query was formulated differently, but because the access control layer silently applies the appropriate organizational scope. This property is essential for multi-tenant deployments where the same conversational interface serves users across different organizations.

3.5 Data ingestion

A key design requirement is that the flexible retrieval layer should accommodate data from heterogeneous sources without requiring source-specific query paths. The ingestion architecture achieves this through a convergent design: multiple source-specific ingestion paths feed into a single CAS layer, normalizing records to a common format in the process.

The framework supports three ingestion paths:

1. Real-time message-based ingestion: physical IoT devices publish sensor readings to a message broker at regular intervals. A gateway daemon subscribes to these messages, optionally decrypts the payload, augments it with ingestion metadata, and writes it into the CAS.
2. Periodic HTTP-based ingestion: edge devices or gateways push sensor data to the ingestion service via HTTP. The service validates that the reporting device is registered, normalizes the payload, and writes it through the access-controlled session.
3. External API synchronization: a dedicated service periodically fetches data from public APIs, converts measurements into the platform's record format, registers the corresponding devices in the Device Registry, and ingests the records through the standard path.

All three paths converge at the CAS layer. After a record is saved, its origin is encoded only in its metadata fields (organization, source type, device identifier), the storage mechanism, content address, and retrieval interface are identical regardless of the source. This convergence is what enables a single query to transparently return results from both private sensors and public data sources.

3.6 Imputation service

IoT sensor deployments routinely experience data gaps caused by hardware faults, connectivity loss, or power interruptions. When a device stops reporting, the CAS faithfully records the absence: no message arrives, so no content hash is generated and no record is stored. While this preserves data integrity, it degrades the completeness of time-series data available to downstream consumers such as the NL query engine and the dashboard. The Imputation Service addresses this gap by periodically detecting missing hourly readings and inserting predicted values into the CAS, maintaining data continuity without compromising the integrity guarantees of content-based addressing.

The service operates on a **detect-predict-reconcile** cycle. A background scheduler runs at the end of each hour and, for every active device and metric pair, checks whether a measured record exists for the last completed hour. If no measurement is found, the service selects an imputation method, predicts the missing value, and stores it in the CAS as an imputed record (tagged with `source="imputed"` and the method used). When a real measurement subsequently arrives for an hour that was already predicted, the service reconciles the data: the imputed record is replaced by the real measurement, previously predicted values are corrected via cubic spline interpolation

through the known anchor points, and any predictions made after the arrival point are discarded and re-predicted with the benefit of the new real value.

The method selection follows an adaptive strategy based on two factors: the availability of correlated neighboring sensors within a configurable geographic radius, and the number of consecutive hours already imputed for the same device and metric. When neighboring sensors with high Pearson correlation ($r > 0.8$) are available, the service uses *spatial regression*, fitting a linear model from the neighbors' readings to the target device. When correlation is low and the gap is short (six or fewer consecutive hours), an *autoregressive* (AR) model extrapolates from the device's own recent history. When correlation is low and the gap exceeds six consecutive hours, the service escalates to a *spatial neural network* that captures non-linear relationships between the target and its neighbors. If no neighboring sensors exist, the AR model is used regardless of gap length as the only available option.

The three imputation methods employed (AR modeling, spatial regression, and spatial neural network) are well-established techniques in environmental sensor data analysis; the service's contribution lies in their adaptive composition and integration into the content-addressable pipeline with automatic reconciliation against late-arriving measurements, rather than in the methods themselves.

This mechanism balances accuracy against data availability: simple gaps are filled with lightweight temporal models, while extended outages leverage spatial information from the surrounding sensor network. All imputed records are stored through the standard CAS path and are therefore subject to the same content-addressing, deduplication, and RBAC enforcement as measured data. The `source` field distinguishes imputed from measured records, allowing the NL engine and dashboard to present provenance information to the user.

3.7 Natural Language Query Engine

The NL Query Engine addresses the second major barrier to flexible data access: the requirement for technical query expertise. Content-based addressing and predicate queries provide a powerful retrieval mechanism, but formulating predicates requires knowledge of the record schema, field names, value ranges, and temporal formats. The NL engine removes this barrier by providing a conversational interface where users express queries in natural language.

Rather than translating natural language to a database query language (the text-to-SQL paradigm), the engine takes advantage of the function-calling capability of Large Language Models. The LLM is presented with a structured tool schema that describes the available query parameters: metric type, device identifier, organization, city, country, value range, time window, geographic coordinates, and search radius; all optional in any combination. The model interprets the user's question and responds with a structured tool call specifying the relevant parameters.

This approach offers several advantages over text-to-SQL. The LLM's output is limited to a well-defined parameter schema, preventing invalid or malicious queries [5]. Additional processing (city-to-device link, geographic proximity filtering, RBAC scoping) can be applied between the LLM's output and the actual data retrieval, without the LLM needing to model these transformations. The same engine works

with any query backend, as the tool call is an intermediate representation that is independent of the storage layer.

A known limitation of language models is unreliable arithmetic, particularly when computing Unix timestamps from relative expressions such as "last week" or "since March" [15]. The engine addresses this by pre-computing a temporal reference table at query time, containing timestamp boundaries for each month, each day of the current week, and common relative offsets, and injecting it into the system prompt. The LLM selects from pre-computed values rather than performing arithmetic, reducing temporal reasoning errors.

To overcome model limitations, the engine implements graduated fallback strategies. If the model produces a tool call in the expected structured format, it is executed directly. If the model outputs the tool call as plain text, a parser extracts and executes it. If the model fails to produce any tool call, the engine infers query parameters directly from the question text using pattern matching. These fallbacks ensure that the system remains functional across the spectrum from high-capability cloud models to resource-constrained local models, degrading gracefully rather than failing entirely.

The engine supports multi-turn interactions by accepting conversation history, enabling follow-up questions that reference prior context (e.g., "and what about humidity?" after a temperature query).

3.8 User interface

The user interface provides two complementary interaction modes, both operating through the same access-controlled session. The dashboard mode is a structured exploration view where users select a device, inspect its metadata, and vizualize time-series data. The device list is dynamically populated based on the user's permissions: unrestricted users see all devices, while organization-scoped users see only their own devices plus public sources. The conversational mode has a chat interface where users type natural language questions and receive both a natural language summary and the underlying query results. This mode maintains conversation history for follow-up queries and exposes the engine's tool calls for transparency.

Both modes enforce identical access control, ensuring consistency: the same user sees the same data whether browsing the dashboard or asking a question in natural language.

4 Implementation details

This section describes the concrete implementation of the framework presented in Section 3. All components are implemented in Python 3.12 and are designed for hardware with reduced resources, including single-board computers such as the Raspberry Pi.

4.1 Content-Addressable Storage

The CAS is implemented as a dedicated module backed by an SQLite database operating in Write-Ahead Logging (WAL) mode, which enables concurrent read access during

write operations. This property is particularly relevant when the Ingestion Service writes records while the NL engine or dashboard queries the store concurrently.

When a record is ingested, it is first serialized into a canonical JSON representation with lexicographically sorted keys and compact formatting. This deterministic serialization ensures that semantically identical records always produce the same byte sequence, regardless of key insertion order in the originating source. The SHA-256 digest of this byte sequence serves as the record's content address, a 64-character hexadecimal string that uniquely identifies the record.

Each record is stored under its content hash as the primary key. The *device_id* and *timestamp* fields are extracted from the JSON payload into dedicated indexed columns to support efficient device-scoped and temporal lookups, while the complete JSON record is retained in a *payload* column. Timestamps are stored as 64-bit IEEE 754 floating-point Unix epoch values with sub-second precision.

The storage operation uses an insert-or-ignore strategy: if a record with the same content hash already exists, the write is silently discarded. This provides automatic, zero-configuration deduplication, as the deterministic content-to-address mapping and the primary key constraint are sufficient without counters, bloom filters, or application-level duplicate detection.

Content-based queries retrieve stored payloads, deserialize each from JSON, and apply a caller-supplied filter function. Records that lack the queried field are silently skipped rather than raising errors, providing schema-tolerant filtering across heterogeneous records from different sources.

4.2 Device Registry

The Device Registry operates on the same SQLite database as the CAS, ensuring transactional consistency between record storage and device metadata. Each device entry stores geographic coordinates as decimal-degree latitude and longitude, human-readable location identifiers (*city*, *country*) used by the NL engine to resolve location-based queries, a *source* field distinguishing private IoT devices from those imported via external APIs, and a last-modification timestamp.

Device metadata is managed through an upsert operation: inserting a device that already exists updates its location and organization metadata without requiring an explicit delete, supporting device relocation or metadata correction. A junction table implements the many-to-many relationship between devices and organizations, allowing a single device to be shared across multiple monitoring projects and enabling organization-scoped queries to determine which devices belong to a given organization.

When a query specifies a city or country name, the registry returns the set of device identifiers matching that location. When a query specifies coordinates and a search radius, the Haversine formula computes the great-circle distance between the query centre and each device's registered coordinates, using the Earth's mean radius of 6,371 km. Only devices within the specified radius contribute records to the query result.

4.3 User Service

The User Service is implemented as an HTTP server using the FastAPI framework. It maintains its own tables within the shared SQLite database and exposes a RESTful API consumed by the RBAC layer.

The *users* table stores credentials using a salted hash scheme: each user record contains a random 32-byte salt and a hash produced by the PBKDF2-SHA256 key derivation function with 200,000 iterations, following current OWASP guidelines for password storage. The *role* column stores a convenience label (*admin* or *user*), while the *permissions* column stores the actual permission set as a JSON-encoded text array, this is the authoritative source for access control decisions.

The *organization* column associates each user with a primary organization, and the *device_id* column associates the user with a device (used as the default device context in the UI). Both fields are nullable to support administrative accounts that are not organization-specific.

The *org_users* junction table enables users to belong to multiple organizations. Each membership includes an *org_role* column with values *viewer* or *editor*, which further constrains write and delete operations within that organization. This supports scenarios where a user participates in several monitoring projects with different privilege levels.

The *organizations* table serves as the authoritative registry of organization identifiers. The *created_at* column records the registration time as a Unix timestamp in REAL format. This table is the parent in the foreign-key relationships to both *org_users* and *org_devices*.

The User Service exposes endpoints for authentication (login, bootstrap of the initial administrator account), authorization (per-operation permission validation), and user management (create, list, delete accounts). The authorization endpoint accepts a username, a requested permission, and optionally a target resource organization; it returns an allow or deny decision based on the user's stored permission set and organization membership.

To realize RBAC session integration, the client-side RBAC layer communicates with the User Service through an HTTP client with an 8-second request timeout. Every data operation (read, write, delete, device listing, user management) triggers an authorization request to the User Service before proceeding. The session wraps the user's identity context (username, role, organization, device, and permission set) and intercepts all operations to enforce access control transparently. For users with *read_own* permission, the session adds query predicates with an organization filter that restricts results to the user's own organization plus public data sources (organization *openaq*).

4.4 Data ingestion

The framework implements three ingestion paths, all converging at the CAS layer.

The **MQTT** connector is configured to run continuously in the background on a Raspberry Pi. It uses the Paho MQTT client library to subscribe to sensor topics using a wildcard pattern (*sensors/#*), receiving messages from all subtopics. Incoming

payloads are decrypted via Fernet symmetric encryption, which ensures both confidentiality and integrity by combining AES-128 and HMAC-SHA256 within a fail-safe implementation. During ingestion, the system appends metadata to each record, such as the MQTT topic and the time of receipt.

The second path is **HTTP**. The Ingestion Service is implemented as a FastAPI server, exposing two endpoints. The registration endpoint accepts device metadata (identifier, geographic coordinates, organization) and creates or updates the device in the Device Registry. The ingestion endpoint accepts a sensor payload, validates that the reporting device is registered (configurable to allow unregistered devices during development). Then it normalizes the record to ensure *device_id* and *timestamp* fields are present and writes it through the RBAC session using a privileged service account with *write* permission.

The last ingestion way is through the **OpenAQ Service**. It fetches air quality measurements from the OpenAQ v3 public API [13] using the official Python client. Each measurement is converted into a CBA-compatible record tagged with organization: "openaq" and source: "openaq". The corresponding sensor is registered in the Device Registry with its geographic coordinates, city, and country extracted from the API metadata. A command-line synchronization tool supports both targeted sensor fetching (by sensor identifier) and geographic discovery, where sensors are located by specifying a latitude, longitude, and search radius. Because ingestion passes through the standard RBAC-gated path, public records undergo the same content-addressable storage and deduplication processes as private sensor data.

4.5 Imputation service

The Imputation Service is implemented as a Python module comprising three components: the core service class, a periodic scheduler, and a utility library for spatial and temporal computations.

The **scheduler** runs as a daemon thread and checks for missing values each hour. Each time, it authenticates via the RBAC session, retrieves all registered devices from the Device Registry, and invokes the core service for each device-metric pair.

The **core service** maintains a dictionary that tracks the number of consecutive imputed hours per device-metric pair. For each check, it queries the CAS for measured records in the target hour. If a measured record exists, the consecutive counter resets to zero. If not, the service gathers training data from a look-back window that expands proportionally with the gap length (minimum 24 hours, up to three times the consecutive count), discovers neighboring sensors within 20 km using the Haversine formula, and computes Pearson correlations between the target and each neighbor's hourly means.

Three prediction methods are implemented. The *autoregressive* (AR) method uses Linear Regression on three lagged features, predicting recursively when multiple values are needed. The *spatial regression* method fits a linear model using the five most correlated neighbors' concurrent readings as features. The *spatial neural network* uses a TensorFlow `Sequential` model with three dense layers (64, 32, 16 neurons) and

dropout regularization, trained on the same neighbor features. Method selection follows the decision matrix described in Section 3.6, with escalation from AR to spatial NN after six consecutive imputed hours.

The **reconciliation** procedure is triggered by the Ingestion Service when a real measurement arrives. It proceeds in three steps:

1. *Replace*: imputed records for the arrival hour are deleted from the CAS; the real measurement has already been stored by the Ingestion Service.
2. *Spline correct*: a cubic spline is fitted through all measured anchor points in a 48-hour window. Imputed records falling between anchors are re-interpolated.
3. *Re-predict*: imputed records after the arrival hour (if they exist) are deleted and re-predicted sequentially, with the consecutive counter reset to zero. This restarts the method escalation from AR, since the new real value now falls within the training window.

4.6 IoT hardware

The physical IoT node is built on an ESP8266 (NodeMCU) microcontroller programmed in Arduino C++. The device interfaces with three sensors: a DHT11 digital sensor providing temperature in degrees Celsius and relative humidity in percent, and an MQ7 analog sensor measuring carbon monoxide concentration. Readings are sampled and published every 30 seconds as a JSON payload to the MQTT broker, authenticated with username and password credentials. Wall-clock time is obtained via NTP synchronisation with pool.ntp.org to ensure accurate timestamps.

For development and evaluation purposes, a software simulator generates synthetic sinusoidal sensor data at 5-second intervals, posting to the Ingestion Service over HTTP. This allows full end-to-end system testing, including NL query evaluation, without requiring physical hardware.

4.7 Natural Language Query Engine

The NL engine is implemented as a Python class that communicates with any OpenAI-compatible API endpoint through the official OpenAI client library. The same class supports both local inference via Ollama and cloud-based inference via OpenAI, selectable through configuration parameters. Its capabilities are summarized in Fig. 2.

Regarding the tool schema, two tools are registered using the function-calling interface. The *query_records* tool accepts twelve optional, fully combinable parameters: metric type, device identifier, organization name, city, country, minimum and maximum sensor value, minimum and maximum timestamp, latitude, longitude, and search diameter in kilometers. The *list_devices* tool accepts no parameters and returns all registered device identifiers that the user has access to.

At query time, the engine constructs a system prompt containing three elements: a description of the record schema and field semantics, instructions for correct tool-call formatting (including explicit guidance on using JSON null for absent values), and a pre-computed temporal reference table. The temporal table contains Unix timestamp boundaries for each month of the current year, each day of the current week, and common relative offsets (1 hour, 6 hours, 24 hours, 7 days, and 30 days before the current

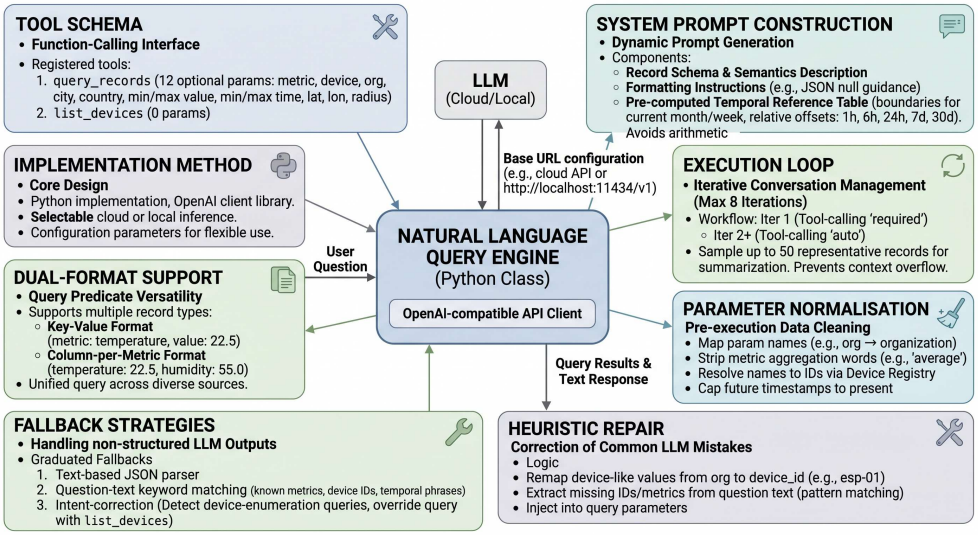


Fig. 2: Natural Language Query Engine capabilities

time). The model is instructed to select values from this table rather than computing timestamps arithmetically. This strategy is used to mitigate known weaknesses in language model arithmetic reasoning.

The engine operates an iterative loop, capped at eight iterations, that manages the conversation between the user's question and the LLM's tool calls. On the first iteration, the tool-calling mode is set to required, forcing the model to produce at least one tool call before generating a text response. This prevents the model from hallucinating an answer without consulting the data store. On subsequent iterations, the mode is set to auto, allowing the model to either issue additional tool calls or produce a final summary. To prevent context overflow when queries return large result sets, the engine samples up to 50 representative records proportionally across devices before sending them to the LLM for summarisation.

Before executing a tool call, the engine applies several normalization steps: mapping alternative parameter names (e.g., org to organization) produced by different models; stripping aggregation words from metric names (e.g., transforming "average temperature" to "temperature") to prevent semantic leakage from the question into the filter; resolving city and country names to device identifier sets via the Device Registry; and limiting future-dated timestamps to the current time, ensuring that queries are not referring to the future.

The engine applies heuristic corrections for common model mistakes: if the organization parameter contains a value that resembles a device identifier (e.g., esp-01), it is remapped to the *device_id* parameter; if the model omitted a device identifier or metric that is clearly mentioned in the question text, the engine extracts it via pattern matching and injects it into the query parameters.

The engines presents multiple fallback strategies. Three fallbacks handle models that do not use the structured function-calling interface. First, a text-based parser

detects tool calls embedded as JSON objects in the model's plain-text response, including resolution of inline arithmetic expressions in timestamp values. Second, if no tool call is detected, the engine constructs query parameters directly from the question text by matching against known metric names, device identifier patterns, city names, and temporal phrases. Third, an intent-correction mechanism detects device-enumeration questions (identified by keywords such as "which devices" or "list devices") and overrides an incorrectly chosen *query_records* call with *list_devices*.

The query predicate supports two record formats: a key-value format where the metric type and value are stored in separate fields (metric: "temperature", value: 22.5), and a column-per-metric format where each metric is a field name (*temperature: 22.5*, *humidity: 55.0*). This allows the engine to query records ingested from different sources without requiring format unification at ingestion time.

4.8 User interface

The web interface is implemented using the Streamlit framework. After authentication, the application presents two interaction modes: dashboard mode and conversational mode. The first is only for data visualization.

In the conversational mode a chat interface accepts natural language questions and maintains conversation history in the session state to support follow-up queries. When a user has selected a device in the dashboard and the question does not mention a specific location, the interface automatically enriches the question with the selected device's context. Responses display the LLM's natural language summary alongside expandable sections showing the raw tool calls and returned records for transparency.

A sidebar is used to display connected user details (name, role, organization, permissions) and refresh rate configuration. The LLM backend, model name, and API key is also configurable through that component without restarting the application.

5 Evaluation and experimental results

This section presents a quantitative evaluation of the framework across its three core subsystems: content-based addressing performance, role-based access control correctness, and natural-language query effectiveness. All experiments were conducted on a commodity laptop (Intel Core i7, 16 GB RAM, NVMe SSD) running Python 3.12 with SQLite in WAL mode.

5.1 Content-Based Addressing performance

We benchmarked the CAS layer at three data scales (1K, 10K, and 100K unique records) to characterize write throughput, read latency, predicate query cost, and deduplication correctness. At each scale, 20% duplicate records were injected to stress the content-addressing mechanism.

Write throughput (records per second) from Table 1 remains in the same range across all three scales. We can make the same affirmation regarding the average write latency, inserting one record takes less than 2ms. This near-constant performance is attributable to SQLite's B-tree indexing on the SHA-256 address column and the

Table 1: CAS write throughput and deduplication

Scale	Records written	Throughput (rec/s)	Avg write latency (ms)	Duplicates rejected	Dedup correct
1K	1,200	540.0	1.85	200	✓
10K	12,000	533.3	1.88	2,000	✓
100K	120,000	523.2	1.91	20,000	✓

INSERT OR IGNORE semantics that make duplicate rejection an O(log n) index lookup rather than a full-table scan. Deduplication was verified to be correct at all scales: exactly 16.67% of writes were rejected as expected, and the rest up to 20% duplicate records were silently ignored.

Hash-based retrieval (Table 2) exhibits sub-millisecond latency at all scales. Predicate and range queries scale linearly with store size, as expected for full-table scans with in-memory predicate evaluation. At 100K records, predicate queries complete in under 300 ms, acceptable for interactive dashboard refresh scenarios. For deployments requiring sub-second response at larger scales, additional indexes can be defined without modifying the CAS contract.

Table 2: CAS read latency

Scale	Hash lookup avg (ms)	Hash lookup p95 (ms)	Predicate query avg (ms)	Range query avg (ms)
1K	0.006	0.007	2.72	2.85
10K	0.009	0.013	31.12	32.57
100K	0.017	0.031	296.8	335.9

Regarding the storage overhead, the SQLite database incurs a 2.6–2.76× overhead relative to raw JSON payload size, which covers the B-tree index, WAL journal, and per-record metadata. This overhead is constant across scales and represents a reasonable trade-off for O(log n) address lookups and ACID guarantees.

5.2 Role-Based Access Control correctness

RBAC correctness was validated through an exhaustive permission matrix covering all combinations of roles, permissions, and resource scopes. The test suite exercises 40 distinct access scenarios spanning four role types:

- **Admin**: full permissions (*write*, *delete*, *read_all*, *list_devices*, *manage_users*)
- **Standard user**: organization-scoped read access (read_own)
- **Elevated user**: cross-organization read (*read_all*, *list_devices*) without write
- **Write-capable user**: ingestion permission without administrative access

Each test case specifies a user profile, an operation, a target resource, and the expected outcome (allow or deny). For example, a standard user with *read_own* permission in organization B queries records belonging to organization A. It is expected that the query succeeds but returns an empty result set, as the organization filter

silently excludes non-matching records. Then the same user queries records with organization "openaq". We expect that results are returned, because public data sources remain visible to all authenticated users. Another scenario is when a write-capable user attempts to delete a record, and we expect a `PermissionError`, as the user holds *write* but not *delete* permission. For an elevated user with *read_all* and *list_devices* attempts to create a new user account the result is a rejection, as *manage_users* is not in the permission set. The full matrix covers five user profiles across eight operation types (write, delete, read per organization, list devices, manage users), yielding 40 distinct scenarios that collectively verify both positive authorization and negative enforcement.

All 40 test cases passed (100% correctness). The RBAC layer correctly enforces:

- Permission-based gating for all six permission types
- Organization-scoped isolation for *read_own* users
- Public data visibility regardless of user organization
- Proper delegation semantics (only *manage_users* holders can create/delete accounts)

5.3 Natural-Language query evaluation

The NL query engine translates user questions into structured tool calls against the CAS store. We evaluated this component using a benchmark of 100 natural-language questions spanning six different categories of questions (metric filters, temporal constraints, geospatial queries, device listings, value ranges, and composite multi-constraint queries). Each question has a ground-truth tool call with expected parameters. The evaluation store was populated with a controlled dataset designed to cover the query dimensions supported by the framework. Eleven sensor records were ingested across three organizations (*org-a*, *org-b*, *org-c*) and three devices (*esp-01*, *esp-02*, *esp-03*), spanning four metric types (temperature, humidity, pressure, CO_2). Records were assigned timestamps spread across the preceding 48 hours at two-hour intervals, ensuring that temporal filter queries return non-empty results. Three device locations were registered in the Device Registry, all within Cluj-Napoca, Romania, at slightly different coordinates, enabling geographic proximity queries. Two user accounts were configured in addition to the administrator: one with *read_all* and *list_devices* permissions (simulating an unrestricted analyst) and one with *read_own* permission scoped to a single organization (simulating a restricted operator). The benchmark questions specify which user account to use, so that both permission-scoped and unrestricted queries are exercised. Each question passes through the full pipeline (system prompt construction, LLM inference, tool-call parsing, parameter normalization, RBAC-scoped query execution, and result summarization) exactly as it would through the conversational UI.

Table 3 presents illustrative examples from each category, showing the natural-language question, the ground-truth parameters against which the model's output is evaluated, and the specific capability being tested. For most category the expected tool call is 'query_records'

To assess model sensitivity, we compared two locally-hosted LLMs of different parameter counts and architectural structure: Llama 3.2 (3B parameters, developed

Table 3: Illustrative benchmark questions from each category

Category	Example question	Expected parameters	What it tests
Metric filter	"Are there any pressure readings?"	`metric`=pressure	Single-attribute content lookup; the model must extract the correct metric from a colloquial existence check.
Temporal constraint	"What data do we have from last Monday?"	`min_timestamp`, `max_timestamp`	Day-of-week temporal resolution; the model must select pre-computed timestamp boundaries rather than perform date arithmetic.
Geospatial query	"What is the humidity in Cluj-Napoca?"	`metric`=humidity, `city`=Cluj-Napoca	City-name to device resolution; tests whether the model maps a place name to the geographic filter parameter.
Device listing	"What devices are in my organization?"	no parameter; tool call: `list_devices`	API endpoint selection; the model must choose `list_devices` instead of `query_records`.
Value range	"Show pressure data between 1010 and 1015 hPa."	`metric`=pressure, `min_value`=1010, `max_value`=1015	Implicit metric inference from the unit (hPa) combined with dual-bound numeric range extraction.
Composite multi-constraint	"Humidity between 50 and 70 % from org-a in the last week."	`metric`=humidity, `organization`=org-a, `min_value`=50, `max_value`=70, `min_timestamp`	Five-dimensional parameter extraction spanning metric, organization, value range, and temporal filter; the most challenging category.

by Meta) and Qwen 2.5 (7B parameters, developed by Alibaba). Both models were served via Ollama using identical system prompts and tool-calling schemas, isolating the effect of model capacity on structured output generation.

Each metric in Table 4 captures a different aspect of structured output quality. *Tool selection accuracy* is the fraction of queries where the model selected the correct API endpoint, regardless of parameter correctness. *Exact match* is the fraction of queries where the model produced both the correct tool and the exact set of parameters matching the ground truth; a question that selects the right tool but omits a temporal filter or adds an unnecessary field scores 1.0 on tool accuracy but 0.0 on exact match.

Precision and *recall* quantify partial-credit performance for queries that select the right tool but produce an incomplete or over-specified parameter set. Crucially, both metrics are computed *per parameter within each question*, not per question as a binary score. For a given question, let M denote the set of model-generated parameters, and G the set of ground-truth parameters. Per-question precision is $|M \cap G| \,/\, |M|$ (the fraction of generated parameters that are correct, penalizing hallucinated parameters), and per-question recall is $|M \cap G| \,/\, |G|$ (the fraction of expected parameters that the model extracted, penalizing omissions). The reported values are the averages of these per-question scores across all 100 questions. This formulation means that a question with,

for example, three generated parameters of which two are correct yields a precision of $2/3 \approx 0.67$, rather than a binary 0 or 1. The F1 score is the harmonic mean of averaged precision and recall.

Qwen 2.5 achieves a higher tool selection accuracy than Llama 3.2 (96% vs. 93%), indicating that while even a 3B-parameter model reliably identifies the correct API endpoint in most cases, the larger model is more robust across diverse phrasings. The differentiation is more pronounced in parameter extraction quality: Qwen 2.5 achieves a 12-percentage-point improvement in F1 score (0.786 vs. 0.670), driven primarily by superior recall (0.853 vs. 0.672). This suggests that the larger model more consistently extracts all relevant constraints from complex queries, whereas Llama frequently omits parameters such as temporal bounds or value ranges.

However, this accuracy improvement comes at a 2.2× latency cost (50.72 s vs. 22.94 s per query). This trade-off is characteristic of edge-deployed LLM inference on consumer hardware without GPU acceleration. In production deployments where interactive response is prioritized, the smaller model offers an attractive accuracy-latency balance; for batch or asynchronous analytics, the larger model's superior parameter extraction may justify the additional inference time.

Table 4: NL Query Engine multi-model comparison

Metric	**Llama 3.2 (3B)**	**Qwen 2.5 (7B)**
Tool selection accuracy	93%	96%
Exact match rate	29%	36%
Parameter precision	0.668	0.729
Parameter recall	0.672	0.853
F1 Score	0.670	0.786
Avg latency (s)	22.94	50.72

Table 5 breaks down the metrics defined above across six representative categories. The same per-parameter precision and recall formulation applies: within each category, scores are averaged over the n questions in that group.

From the category breakdown we observed the following patterns. Temporal reasoning remains the most challenging query type: although the expanded benchmark shows non-zero exact match rates (Llama 0.28, Qwen 0.17), both models still struggle to translate informal phrases such as "past 24 hours" into precise Unix timestamps. Geospatial and metric filters are well-served by both models, with Qwen achieving 0.81 precision and 0.93 recall on geographic queries. Value-range queries expose a sharp capability gap: Llama achieves 0% exact match despite 1.0 tool accuracy, indicating that it consistently selects the correct endpoint but fails to extract dual-bound numeric constraints, whereas Qwen reaches 60% exact match on the same questions. Composite multi-constraint queries ($n = 40$) show both models at 97% tool accuracy, but Qwen's recall advantage (0.89 vs. 0.77) confirms that larger parameter counts improve multi-dimensional parameter extraction. Device listing is the category with the most pronounced performance divergence: Llama achieves only 25% tool selection accuracy compared to Qwen's 62%.

Table 5: Category-level performance. L = Llama 3.2, Q = Qwen 2.5.

Category	n	Tool acc. (L/Q)	Exact match (L/Q)	Precision (L/Q)	Recall (L/Q)
Metric filter	10	1.0 / 1.0	0.40 / 0.50	0.67 / 0.68	0.90 / 0.90
Temporal constraint	18	1.0 / 1.0	0.28 / 0.17	0.61 / 0.56	0.68 / 0.75
Geospatial query	14	1.0 / 1.0	0.29 / 0.43	0.76 / 0.81	0.67 / 0.93
Device listing	8	0.25 / 0.62	0.25 / 0.62	0.25 / 0.62	0.25 / 0.62
Value range	10	1.0 / 1.0	0.0 / 0.60	0.73 / 0.84	0.37 / 0.90
Composite multi-constraint	40	0.97 / 0.97	0.35 / 0.28	0.73 / 0.78	0.77 / 0.89

Manual inspection reveals that Llama systematically misclassifies device-enumeration questions (e.g., "List all connected devices", "What devices are available?") as data queries. This suggests that the smaller model lacks sufficient capacity to distinguish meta-queries about the device registry from content queries against the data store when both share overlapping vocabulary.

6 Limitations and future work

This section examines the boundaries of the current framework and identifies directions for extending it.

6.1 Limitations

The evaluation in Section 5 revealed several performance ceilings and design trade-offs that constrain the framework's applicability in its current form. These limitations fall into three categories: **storage-layer scalability**, **NL query accuracy**, and **deployment constraints**.

For the first category we identified concerns regarding linear-cost predicate queries, single-node storage and no encryption at rest.

While hash-based retrieval scales logarithmically, predicate and range queries perform a full-table scan with in-memory filtering. At 100K records, average predicate latency reaches 297 ms, acceptable for dashboard refresh, but at larger scales (e.g., millions of records accumulated over months of continuous monitoring) this approach would exceed interactive latency budgets. The current architecture intentionally avoids secondary indexes on variable payload fields to preserve schema-free flexibility, but this places an upper limit on the performance of content-based queries.

The framework relies on a single SQLite database operating. Although this simplifies deployment and provides ACID guarantees suitable for edge gateways, it prevents horizontal scaling. Concurrent write throughput is limited to a single writer at a time, and the store cannot be replicated across multiple edge nodes without an external synchronization mechanism.

While the framework enforces access control through RBAC and uses PBKDF2-SHA256 for credential hashing, the SQLite database itself is stored unencrypted on disk. Any actor with access to the edge node can read the record payloads directly, bypassing the RBAC layer entirely.

Regarding **Natural-Language query accuracy**, the NL evaluation exposed systematic weaknesses in how locally-hosted models handle certain query types, independent of model size.

Despite the system prompt injecting a precomputed lookup table of Unix timestamps for months, weekdays, and relative offsets, both evaluated models achieve low exact match rates on temporal filter queries (Llama 28%, Qwen 17%), frequently omitting or misinterpreting temporal constraints. This remains the most significant accuracy bottleneck in the NL query pipeline, contributing to overall exact match rates of only 29 and 36%.

The evaluation compared only two locally-hosted open-weight models. The results may not generalize to instruction-tuned variants, closed-source models (e.g., GPT-4o), or models fine-tuned on structured-output tasks. The observed 93 (Llama) and 96% (Qwen) tool selection accuracy and 0.670 and 0.786 F1 scores represent a lower bound for the tool-calling approach; larger or specialized models would likely improve parameter extraction.

The NL benchmark comprises 100 selected questions across 6 categories. While this covers the primary query patterns supported by the framework, it may not capture the full distribution of real user expressions, including ambiguous, adversarial, or domain-specific phrasing. Similarly, the CAS benchmarks used synthetically generated sensor records; performance on real-world payloads with variable schema complexity remains to be validated.

Deployment constraints refer to running the full framework, including LLM inference, on resource-constrained edge hardware introduces practical trade-offs that affect usability.

All LLM inference was performed on CPU without GPU acceleration. Average query latency ranged from 22.94 s (Llama 3.2, 3B) to 50.72 s (Qwen 2.5, 7B), which is unsuitable for real-time interactive use. This limitation is environmental rather than architectural. The same models would respond in under a second on GPU-equipped hardware, but it reflects the practical constraints of resource-limited edge deployments.

6.2 Future work

The limitations identified above suggest several concrete research and engineering directions. A logical first step involves optimizing the underlying storage engine. By introducing secondary indexing, the system could move from linear scanning to logarithmic lookup speeds, significantly cutting query costs without breaking the content-addressing contract. This structural change would be improved by provinding security at the "edge," through at-rest encryption to ensure data remains protected even if a physical node is compromised. To make the user experience highly responsive, these speed and security upgrades must be paired with edge-optimized inference; using quantized models and hardware accelerators to improve natural language response times from several seconds down to the sub-second range.

Another direction would be to improve how the system understands and shares data. The gap between "identifying the right tool" and "generating the perfect query" suggests that the model would benefit from domain-specific fine-tuning. Training a

smaller, specialized model could obtain better results for its temporal reasoning and parameter extraction without requiring more expensive hardware.

One last possible upgrade is a natural extension of the imputation service and would be real-time anomaly detection: incoming readings could be compared against the temporal and spatial baselines already computed for gap-filling and flagged automatically when they deviate beyond a configurable threshold.

7 Conclusions

This paper presented a framework for flexible data access in IoT ecosystems, combining content-based addressing, role-based access control, and LLM-driven natural-language querying into a unified architecture. Rather than treating storage, security, and querying as independent concerns, the framework composes them into a layered design where each mechanism reinforces the others: content addressing provides schema-free and deduplicated storage; RBAC enforces organization-scoped access policies over that store. The imputation service maintains data completeness by filling gaps with adaptive predictions and reconciling them against late-arriving measurements; and the NL query engine translates user intent into structured operations against the access-controlled data layer.

The empirical evaluation validated this design across three dimensions. The CAS layer sustained 523–540 records per second with sub-millisecond hash lookups at scales from 1K to 100K records, confirming that content addressing is a viable storage primitive for edge IoT workloads where data integrity and deduplication matter more than raw throughput. The RBAC layer achieved 100% correctness across an exhaustive 40-case permission matrix, demonstrating that fine-grained, organization-scoped access control can be enforced without sacrificing usability. The NL query engine achieved 93–96% tool selection accuracy across the two evaluated models, showing that even small, locally-hosted LLMs can reliably map natural-language questions to the correct API endpoint. The multi-model comparison further revealed a predictable accuracy–latency trade-off: the larger model improved parameter extraction F1 from 0.670 to 0.786 at a 2.2× latency cost, allowing operators to balance response quality against inference time based on their specific deployment context.

The framework's principal contribution is demonstrating that these access mechanisms (addressable by content, governed by policy, adaptive imputation and natural language querying) can coexist in a single system lightweight enough for edge deployment. Content-based addressing eliminates the need for centralized identifier registries. RBAC decouples data visibility from physical storage location. The imputation service ensures data completeness without manual intervention. Natural-language querying removes the requirement that users understand the underlying data schema. Together, these mechanisms lower the barrier to interacting with heterogeneous IoT data while preserving the security, integrity and completeness properties that production deployments require.

Several limitations remain, most notably the linear cost of predicate queries at scale, the weakness of current open-weight models on temporal reasoning tasks, and the inference latency inherent to CPU-only edge hardware. These limitations define clear

directions for future work, including adaptive secondary indexing, domain-specific model fine-tuning, and extending the imputation baselines toward real-time anomaly detection.

Data availability. Air quality data synchronized from the OpenAQ platform is publicly available via the OpenAQ API (https://docs.openaq.org).

Declarations

Conflict of interest. The authors declare no conflict of interest.

References

[1] Ameer, S., Benson, J., Sandhu, R.: Hybrid approaches (abac and rbac) toward secure access control in smart home iot. IEEE transactions on dependable and secure computing **20**(5), 4032–4051 (2022) https://doi.org/10.1109/tdsc.2022.3216297

[2] Athanasopoulou, A., Fotiou, N., Chatzopoulos, A.: Interacting with iot data spaces using llms and the model context protocol. Sensors **26**(4), 1193 (2026) https://doi.org/10.3390/s26041193

[3] Al-Safi, H., Ibrahim, H., Steenson, P.: Vega: Llm-driven intelligent chatbot platform for internet of things control and development. Sensors **25**(12), 3809 (2025) https://doi.org/10.3390/s25123809

[4] Cao, K., Wang, Z.: Fine-grained access control system for multi-user environments. In: 2024 4th International Signal Processing, Communications and Engineering Management Conference (ISPCEM), pp. 534–538 (2024). https://doi.org/10.1109/ispcem64498.2024.00096 . IEEE

[5] Costa, M., Anwar, M., Mercier, D., Randall, M., Hammad, I.: Enhancing accuracy and maintainability in nuclear plant data retrieval: A function-calling llm approach over nl-to-sql. arXiv preprint arXiv:2506.08757 (2025) https://doi.org/10.48550/arXiv.2506.08757

[6] De La Parte, M.S.E., Wang, Y., Martínez-Ortega, J.-F., Martínez, N.L.: A secure and efficient llm-based system for natural language query-to-sql translation in iot data management. IEEE Access **14**, 28435–28456 (2026) https://doi.org/10.1109/access.2026.3665980

[7] Dudeja, R.K., Singh, A., Bali, R.S., Aujla, G.S.: An optimal content indexing approach for named data networking in software-defined iot system. IET Smart Cities **4**(1), 36–46 (2022) https://doi.org/10.1049/smc2.12023

[8] Fedrecheski, G., De Biase, L.C.C., Calcina-Ccori, P.C., Deus Lopes, R., Zuffo, M.K.: Smartabac: enabling constrained iot devices to make complex policy-based

access control decisions. IEEE Internet of Things Journal **9**(7), 5040–5050 (2021) https://doi.org/10.1109/jiot.2021.3110142

[9] Jiang, X., Tian, Z., Li, K.: A graph-based approach for missing sensor data imputation. IEEE Sensors Journal **21**(20), 23133–23144 (2021) https://doi.org/10.1109/jsen.2021.3106656

[10] Kushwaha, S.: Advancements and challenges in the internet of things (iot): Solutions for scalability, security, energy efficiency, and interoperability. In: 2025 World Conference on Cutting-Edge Science and Technology (WCCEST), pp. 1–6 (2025). https://doi.org/10.1109/wccest66994.2025.11389967 . IEEE

[11] Li, X., Liu, T., Chen, C., Cheng, Q., Zhang, X., Kumar, N.: A lightweight and verifiable access control scheme with constant size ciphertext in edge-computing-assisted iot. IEEE Internet of Things Journal **9**(19), 19227–19237 (2022) https://doi.org/10.1109/jiot.2022.3165576

[12] Molek, C.D., Fronteddu, R., Venable, K.B., Suri, N.: Natural language interaction with databases on edge devices in the internet of battlefield things. In: MILCOM 2025-2025 IEEE Military Communications Conference (MILCOM), pp. 838–843 (2025). https://doi.org/10.1109/milcom64451.2025.11310311 . IEEE

[13] OpenAQ: OpenAQ API Documentation (2026). https://docs.openaq.org/docs/introduction Accessed 2026-05-15

[14] Singh, J., Rani, S., Kumar, V.: Role-based access control (rbac) enabled secure and efficient data processing framework for iot networks. International Journal of Communication Networks and Information Security **16**(2), 91–103 (2024) https://doi.org/10.17762/ijcnis.v16i2.6697

[15] Thukral, S., Kukreja, K., Kavouras, C.: Probing language models for understanding of temporal expressions. In: Proceedings of the Fourth BlackboxNLP Workshop on Analyzing and Interpreting Neural Networks for NLP, pp. 396–406 (2021). https://doi.org/10.18653/v1/2021.blackboxnlp-1.31

[16] Wang, X., Zhang, R.: Learning automata-based content forwarding for information-centric iot. IEEE Transactions on Green Communications and Networking **8**(1), 402–412 (2023) https://doi.org/10.1109/tgcn.2023.3306424

[17] Wang, M., Zhang, Y., Yu, B., Hao, B., Peng, C., Chen, Y., Zhou, W., Gu, J., Zhuang, C., Guo, R., *et al.*: Function calling in large language models: Industrial practices, challenges, and future directions. ACM Computing Surveys **58**(9), 1–37 (2026) https://doi.org/10.1145/3788284

[18] Yu, X., Hu, L., Reichman, B., Chu, D., Chandrupatla, R., Zhang, X., Heck, L., Rosing, T.S.: Sensorchat: Answering qualitative and quantitative questions during long-term multimodal sensor interactions. Proceedings of the ACM on Interactive,

Mobile, Wearable and Ubiquitous Technologies **9**(3), 1–35 (2025) https://doi.org/10.1145/3749496

[19] Zhou, J.: Network security applications in the internet of things. Applied and Computational Engineering **109**, 139–144 (2024) https://doi.org/10.54254/2755-2721/2024.17894

[20] Zhang, Z., Lung, C.-H., Wei, X., Chen, M., Chatterjee, S., Zhang, Z.: In-network caching for icn-based iot (icn-iot): A comprehensive survey. IEEE Internet of Things Journal **10**(16), 14595–14620 (2023) https://doi.org/10.1109/jiot.2023.3274653

[21] Zhang, S., Zhang, W., Liang, W., Jin, W., Li, K.: Storsec: A comprehensive design for securing the distributed iot storage systems. IEEE Transactions on Network and Service Management (2025) https://doi.org/10.1109/tnsm.2025.3597550