Technische Universität München

Chair of Geoinformatics

Automated Classification of IoT Sensors and Registration in Urban Data Catalogs

Master's Thesis for
M.Sc. Information Technologies for the Built Environment

Supervisor: Prof. Dr. Thomas H. Kolbe, M.Sc. Joseph Gitahi

Jeffrey Limnardy
September 15, 2025

Technische Universität München

Chair of Geoinformatics

Automated Classification of IoT Sensors and Registration in Urban Data Catalogs

Master's Thesis for
M.Sc. Information Technologies for the Built Environment

Supervisor: Prof. Dr. Thomas H. Kolbe, M.Sc. Joseph Gitahi

Jeffrey Limnardy
September 15, 2025
Updated version with minor editorial corrections in spelling and grammar
from the original submission

Declaration

I confirm that the research and findings presented in this master's thesis are the result of my own work. Where applicable, all references and sources used have been properly cited, acknowledged and documented.

Munich, September 15, 2025 Jeffrey Limnardy

Abstract

Urban Digital Twins (UDTs) rely on Internet of Things (IoT) sensors as fundamental enablers for bridging the physical and virtual domains, but face substantial challenges in data integration, as well as in interoperability. Current sensor data registration to metadata catalogs relies on time-consuming, human-initiated processes susceptible to errors, resulting in stale entries when services evolve. Urban IoT products with hundreds or thousands of individual sensors require substantive grouping and high-quality metadata to enable discoverability across different user groups like municipal agencies, city planners, and transportation departments.

This thesis introduces WRENCH, an open-source modular end-to-end framework that enables continuous automated registration of IoT sensor data into metadata catalogs utilizing Harvesters, Groupers, MetadataEnrichers, and Catalogers as components. Continuous harvesting is achieved through scheduled pipeline runs and component state management. The key innovation of the framework is KINETIC, a novel clustering algorithm designed explicitly for grouping IoT devices that leverages keyword extraction, co-occurrence network analysis, and community detection in identifying thematic groups of devices. Unlike conventional topic modeling techniques that fail with heterogeneous, unbalanced sensor data, KINETIC can identify underrepresented clusters without compromising semantic coherence.

Evaluation on the Hamburg, Osnabrück and Munich datasets demonstrates KINETIC's superior performance compared to traditional methods like LDA, with significantly better clustering quality metrics (Normalized Mutual Information, Homogeneity, Completeness, and V-Measure). Large Language Model features are built into the system to automatically create detailed, descriptive titles and descriptions that manage vocabulary differences between stakeholder groups. WRENCH supports multilingual data processing, provides template pipelines for rapid deployment, and provides ongoing automated updates to keep current metadata catalog entries.

The framework addresses central challenges in urban IoT metadata management and facilitates the growing utilization of digital twinning technologies. Released as an open-source framework, WRENCH provides a foundation for scalable automation across heterogeneous sensor ecosystems and facilitates further developments in Urban Digital Twin implementations.

Acronyms

API	Application Programming Interface
FROST	Fraunhofer Opensource SensorThings-Server
IoT	Internet of Things
KG	Knowledge Graph
KINETIC	Keyword Informed Network Enhanced Topical Intelligence Clustering
LDA	Latent Dirichlet Allocation
LLM	Large Language Model
NMI	Normalized Mutual Information
NTM	Neural Topic Modeling
OGC	Open Geospatial Consortium
SDK	Software Development Kit
TF-IDF	Term Frequency - Inverse Document Frequency
UDT	Urban Digital Twin
WRENCH	Workflow for Registration and Enrichment of Sensor Data

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

Urban Digital Twins (UDTs) provide virtual models of physical environments by integrating diverse data sources, such as 3D city models, sensors, and simulation data. These digital twins significantly improve urban planning and decision-making, including environmental monitoring, transportation, mobility, building management, energy and utility monitoring, and community engagement.

Internet of Things (IoT) bridges the physical and virtual realm of digital twins (DTs). It represents a key enabler for DTs, supporting identification, modeling, connectivity, smartness, cyber-physicality, and usability [9]. State-of-the-art technologies such as IoT, cloud computing, big data analytics, and artificial intelligence have greatly stimulated the development of digital twins with "feedback loops in which physical processes affect cyber parts and vice versa" [10]. The data obtained from tangible items, sensors, IoT devices, and various origins form the fundamental basis of digital twins, providing details about the present condition, actions, and efficacy of the tangible entity [11].

The implementation of UDTs faces significant challenges in data integration and interoperability. Recent studies highlight "main hurdles like gathering data, connecting systems, handling vast amount of information, and making different technologies work together" [12].

Furthermore, IoT data discoverability is considered as one of the most critical yet underaddressed challenges in modern digital transformation, with 74-75% of IoT initiatives failing largely due to data management issues [13]. The complexity stems from IoT data's unique characteristics. IoT generates continuous, heterogeneous streams from distributed devices requiring contextual metadata that traditional data catalogs cannot effectively manage [14]. This creates a situation where organizations collect large amounts of data but are unable to utilize it.

Contextual dependency is a unique challenge in cataloging IoT data. As researchers note, "unlike the general Internet, searching IoT sensor data is not possible without metadata" because sensor readings are "just numbers that depend on metadata to provide context and semantics" [15]. A sensor reading of "10" is meaning-

less without knowing the sensor type, measurement unit, and environmental context. This contextual richness requires more sophisticated metadata management approaches rather than traditional database schemas.

Manual metadata management represents the most significant limitation in current approaches. Tools like Apache Atlas, AWS Glue Data Catalog, and Azure Purview require human intervention for metadata creation, tagging, and classification [16]. This manual process becomes impossible at IoT scale where thousands of devices are considered with constantly new devices being added. Resource-intensive cataloging creates bottlenecks that prevent timely data discovery and analysis.

IoT data discoverability is a central paradigm which requires new solutions in the area of metadata management, semantic understanding, and distributed system architecture. AI automation, graph-based modeling, edge computing, and standardized semantic structures come together to provide us with unprecedented opportunities to tackle these challenges. Those that are implementing these new solutions are already realizing improvements in operational efficiency, reduced manual intervention, and greater ability to derive actionable insights out of their IoT environments. Success requires going beyond traditional approaches to smart, self-service solutions able to accommodate IoT's unique nature while unleashing the revolutionary business value data discoverability makes attainable [17].

Despite these interoperability and discoverability challenges, the potential of UDTs continues to drive innovation, with 85 percent of IoT platforms expected to contain some form of digital twinning by 2025 [18]. This growing adoption highlights the critical importance of robust frameworks that can effectively manage the complexity of integrating diverse IoT data sources while maintaining the real-time synchronization essential for effective urban digital twin operations.

IoT Data Sources

There is a wide range of IoT data sources available in urban environments. In the context of urban sensors, common examples span multiple domains, including weather and air quality sensors in the environmental domain, traffic and mobility sensors in the transportation domain, and energy consumption and production sensors in the energy domain, among many others. These data sources are

often heterogeneous, with different data models, formats, and protocols. This heterogeneity poses significant challenges for integrating these diverse data sources into a unified metadata catalog.

An example of a standardized data model for IoT sensors is the OGC SensorThings API, which provides a RESTful interface for accessing sensor data and metadata. It allows for registering sensors, their observations, and related metadata in a standardized way [2]. Other than that, several other data models and standards are available, such as the FIWARE Smart Data Models [19], which provide a standardized way to represent IoT data across domains. These standards aim to improve interoperability and facilitate the integration of diverse IoT data sources into urban digital twins.

Metadata Catalogs

The diverse dataset required to manage these UDTs can be organized and documented in urban data catalogs. Urban data catalogs provide metadata to these datasets, including the organization responsible for the dataset, what kind of information a dataset holds, and other information [20]. They also facilitate structured data indexing and discovery.

The Smart District Data Infrastructure (SDDI) project develops the SDDI Catalog, a CKAN-extended metadata catalog for urban data, which includes additional features to improve the representation and discoverability of urban data. Some notable features include support for temporal and spatial filtering, as well UI-based relationship management between datasets [20]. The SDDI Catalog provides a standardized framework for metadata management to improve discoverability and usability of urban data. It allows users to search for datasets based on various criteria, such as location, time, and data type, thereby improving the discoverability of urban data resources.

Moreover, the SDDI framework includes a set of predefined categories and metadata fields that are specifically designed for urban data. It provides a set of main categories which define what kind of data an entry represents, such as "Online Service", "Device", "Project", "Online Application", "Geo-object", "Method", "Software", "Digital Twin", and "Project". These categories help users organize and classify urban data in a meaningful and helpful way. For example, maintainers can use the "Online Service" category to

describe IoT services that provide real-time sensor data. The "Device" category can represent individual sensors or sensor groups. In addition to these main categories, the SDDI framework also includes urban topics such as "Mobility", "Energy", and "Environment" to further categorize urban data, enabling a more thematic grouping of urban data, allowing the users to easily find and access the datasets they need based on their interests [20].

Another example is the MetaVer metadata catalog, which uses the InGrid software [21]. MetaVer is a central access point for urban metadata across multiple German federal states, supporting standardized metadata management for urban datasets. Eight German states actively use the platform—Brandenburg, Bremen, Hamburg, Hessen, Mecklenburg-Vorpommern, Saarland, Sachsen, and Sachsen-Anhalt [22].

1.2 Problem Statement

Metadata catalogs contain information about datasets, such as real-time sensor data collected from IoT devices around the city or generated from simulations. They do not store the data itself, but rather give information about the dataset, such as where users can access the data or who the publisher of the data is.

Urban sensor services, particularly implementations of the SensorThings API, are typically registered in metadata catalogs through manual processes requiring significant time and effort. For catalog entries to be valid, they must include comprehensive metadata such as the types of devices available in the service, their spatial and temporal coverage, and the parameters they measure. This manual process requires catalog maintainers to systematically explore entire service endpoints to identify sensor groups and extract relevant metadata information [23]. Individual device groups are typically created as separate entries in the catalog, linked to their origin sensor service to improve discoverability. The process is time-consuming and prone to producing incomplete or inaccurate catalog entries [24]. Moreover, manually maintained entries often become outdated when services evolve by adding new sensors and modifying or removing existing ones, leading to metadata that no longer reflects the current state of the service. Automating this registration process would significantly reduce maintenance overhead while improving the accuracy and completeness of metadata catalog entries.

High-quality metadata is essential for ensuring discoverability in open data ecosystems. Users of metadata catalogs come from diverse backgrounds, including municipal agencies, urban planners, transportation departments, and private-sector firms involved in construction and infrastructure. However, differences in domain-specific terminology can hinder data accessibility. For instance, a traffic sensor specialist searching for "radar sensor" or "inductive loop sensor" may struggle to find relevant data if the entry is labeled only as "vehicle count", which might be more intuitive for policy-makers. This problem shows the need for comprehensive metadata strategies incorporating diverse terminologies to enhance discoverability across user groups.

Another challenge is automatically identifying and grouping related devices within extensive sensor services. Urban IoT services often contain hundreds or thousands of individual sensors that must be organized into meaningful clusters based on their function, location, or measured parameters. Manual identification of these device groupings is labor-intensive and requires domain expertise to recognize which sensors belong together. Without proper device clustering, metadata catalogs risk becoming overwhelmed with individual sensor entries, making it difficult for users to discover relevant sensor groups for their specific needs.

1.3 Research Objective

The primary goal of this thesis is to automate the registration process and enhance the discoverability of sensor data in metadata catalogs by processing the input data from the source, generating feasible device groupings with informative metadata that users can use for searching and filtering purposes. This thesis will propose a well-defined clustering process to create feasible device groups representing the contents of the metadata catalog, and in addition to that, an automated registration pipeline to register data sources into metadata catalogs.

In the theoretical exploration part, the current state of the publicly available IoT services and metadata catalogs will be thoroughly described and analyzed. This comprehensive review will examine the current landscape of IoT data services and platforms, including publicly available FROST servers, other implementations of the OGC SensorThings API, enterprise IoT platforms like FIWARE, and municipal IoT systems. The review also includes an analy-

sis of existing metadata catalogs, such as the SDDI Catalog and MetaVer, to understand their capabilities and limitations in managing urban sensor data. Furthermore, the study will assess how well current services address user needs across different stakeholder groups, evaluate the effectiveness of existing integration approaches, and identify hurdles that prevent seamless metadata management in heterogeneous IoT environments.

The first part of the thesis explores how to automate sensor device registration and their metadata from IoT services, such as the OGC SensorThings API, into an urban metadata catalog. Through this automation, changes in the IoT services will automatically be reflected in the metadata catalog through continuous data harvesting. The latter part explores methods for clustering diverse sensor data and enriching metadata catalog entries with the derived clusters with their generated descriptive metadata (title, description) and extracted metadata (spatial and temporal extent, data point frequency, and others). The practical aspect of the thesis involves a proof-of-concept implementation using sensor data from existing IoT platforms managed by cities.

2 Literature Review

The rapid development of IoT devices and platforms has created a highly fragmented ecosystem characterized by significant interoperability and data integration challenges. As noted by [25], "the challenge of data integration and interoperability in IoT is multifaceted, including technical issues related to the heterogeneity of devices and systems, as well as the need for common standards and protocols to enable seamless communication". This heterogeneity is further compounded by the fact that each IoT platform is developed with its device identification system, making it "challenging to identify each sensor device between heterogeneous IoT platforms owing to the resource request format (e.g., device identifier) varying between platforms" [26].

Several existing frameworks and standards address aspects of IoT data integration and management. Still, significant gaps remain for enabling a fully automated sensor device classification and metadata catalog registration in urban contexts.

Automated Sensor Device Classification

Current methods of sensor classification use ontologies to distinguish between different types of sensors. In [27], the authors explore different ontologies and discover many semantic similarities between them. They can classify sensor data into SSN. In [28], the authors could automatically register and bind sensor data through several steps, including "semantic description of sensors and things as well as their attributes using ontologies".

The work in [1] proposes a classification framework using metadata assisted cascading ensemble classification (MACE) to annotate open IoT data. The MACE framework uses a sequential, ensemble-based approach to add classes as metadata to existing IoT data. Filtering strategies reduce the number of classes passed to the next classifier. Figure 1 shows the described architecture of the MACE framework.

The authors of [29] combine knowledge graph and retrieval augmented generation capabilities of large language models, resulting in a well-performing model in question-answering tasks related to biomedical topics by leveraging a large biomedical KG to add context to prompts.

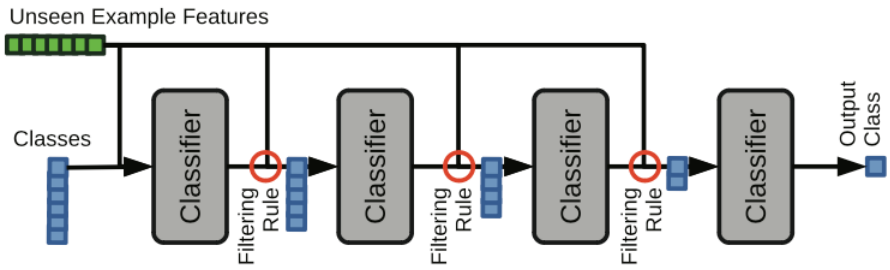

Figure 1: Overview of the MACE Framework [1]

In [30], the authors introduce a novel hierarchical classification framework, utilizing LLMs and text extraction from a document corpus as class and document annotators, allowing the classification model to include hierarchical information, achieving excellent results for the classification with minimal supervision.

Another publication tackles the development of a cross-institutional data catalog framework for FAIRification of environmental research data, with FAIR referring to findability, accessibility, interoperability, and reusability. Their architecture includes a data source-specific Harvester and Ingester, which scans and harvests datasets, one is a harvester and ingester for OGC SensorThings API [31].

Topic modeling is a well-established method in the natural language processing domain. It discovers latent topics available in the data source and clusters different data according to these discovered topics. A very well-known topic modeling algorithm is the Latent Dirichlet Allocation model.

Latent Dirichlet Allocation is a generative probabilistic topic modeling technique that discovers latent thematic structures within document collections. The model employs Dirichlet priors to enforce sparsity, meaning documents typically contain only a few topics and topics use only a subset of the vocabulary, making the discovered topics more interpretable. LDA has been widely applied in text mining, information retrieval, and content analysis to automatically identify coherent topics from large corpora. However, it assumes topics are independent and requires a predetermined specification of the number of topics, which can limit its applicability in some domains [32].

Another more modern approach is BERTopic. BERTopic is a neural topic modeling approach that leverages pretrained trans-

former embeddings (such as BERT) to capture semantic relationships between documents, followed by dimensionality reduction with Uniform Manifold Approximation and Projection (UMAP) [33] and density-based clustering with Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) [34] to identify topic clusters. Unlike traditional methods such as LDA that rely on bag-of-words representations, BERTopic preserves semantic meaning through contextualized embeddings and automatically determines the optimal number of topics, while generating topic representations using class-based TF-IDF to extract the most characteristic words for each cluster. This approach has demonstrated superior performance in capturing coherent and interpretable topics, though it requires more computational resources and can be sensitive to the choice of embedding model and clustering hyperparameters [35].

Recent works in topic modeling for detecting scarce topics in unbalanced short-text datasets introduce a co-occurrence-based topic modeling approach. The authors presented "Topic model based on co-occurrence word networks for unbalanced short text datasets (CWUTM)". A co-occurrence network comprises nodes and edges, where nodes represent words, and edges connecting the nodes represent the number of co-occurrences between the two words. These models prioritize identifying scarce topics in unbalanced short-text datasets, which is particularly relevant for the classification of sensor data, where some classes may have significantly fewer instances than others. Additionally, sensor devices are usually described with short-text descriptions, making this approach suitable for the use case presented in this thesis [36].

Metadata Cataloging and Data Integration

The Smart District Data Infrastructure project proposes an extended catalog service tailored to UDTs, including extended metadata models to better represent urban data registered into the catalog, improving the discoverability and findability of urban data in catalogs. It is implemented by extending the CKAN platform with various features such as temporal and geographical filtering [20].

Several cities and regions have implemented the SDDI Catalog for digital twin initiatives, including the state of Bavaria's TwinBy project [37] and the city of Ingolstadt's SAVeNoW project [38].

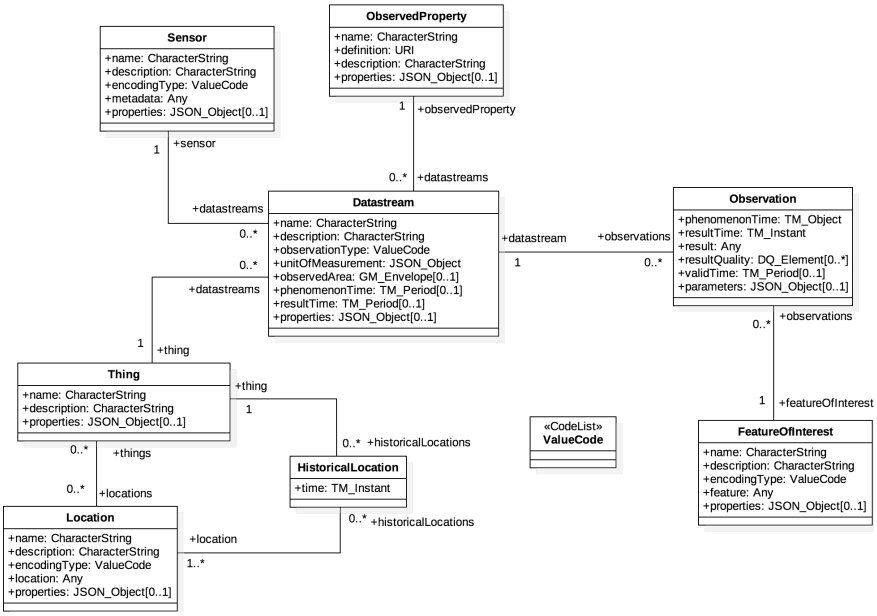

Figure 2: UML diagram of the OGC SensorThings API Entity Data Model [2]

IoT Standards and Interoperability Challenges

The Open Geospatial Consortium has developed the SensorThings API data model to provide a non-proprietary, platform-independent, and perpetually royalty-free Standard. The core objective of the SensorThings API is to enable IoT devices and their data to be easily accessible over the web, facilitating the integration of sensor data from different sources and enabling standardized data exchange. Its two main components—Sensing and Tasking—allow for a comprehensive approach to managing IoT data and actuation capabilities, respectively [2]. Several implementations of the SensorThings API exist, the most popular of which is the FROST Server created by the Fraunhofer-Institut [39].

The SensorThings API defines entities for representing sensors and their observations, including Thing, Observation, Location, HistoricalLocation, ObservedProperty, Datastream, Sensor, and FeatureOfInterest [2]. The specification also establishes querying, filtering, and entity expansion capabilities that compliant services like FROST Servers must implement.

The FIWARE Foundation provides an open-source platform that

aids in developing open-source implementations focusing on smart solutions, digital twins, and data spaces in a wide variety of domains. The foundation has developed the Smart Data Models Initiative, a standardized data model to enhance interoperability between IoT devices and domains. These include predefined schemas for smart cities, agriculture, energy, transportation, monitoring, etc. The schemas make it easier to integrate diverse datasets and improve inter-domain collaboration in the smart-application domain [19].

IoT Standards and Protocols such as the OGC SensorThings API [2] and FIWARE [19] attempt to address interoperability through standardization. These standards provide valuable frameworks for data exchange and enable a more standardized way of representing IoT data. However, the current implementations of these standards often lack comprehensive thematic classification and metadata enrichment capabilities, which are essential for effective urban digital twin management in metadata catalogs.

The OGC SensorThings API, for example, provides a standardized data model to represent sensors, their observations, and other relevant metadata. It was designed to provide a "unified way to interconnect the Internet of Things (IoT) devices, data, and applications over the web" and "manage and retrieve observations and metadata from heterogeneous IoT sensor systems" [2], suggesting a design philosophy that prioritizes broad interoperability over domain-specific optimization. As a standardization effort, the API aims to serve diverse sectors including the smart city and urban digital twin domains. With this domain-agnostic data model, the standard allows organizations across different domains to exchange data consistently and efficiently. Therefore, it does not include any thematic standardization within domains. The standard has neither restrictions nor enforced requirements to categorize devices into specific classes or any thematic domain.

The API provides several features, including the possibility to filter devices based on their "Observed Properties" or "Sensors". Researchers in [3] have attempted to use these filters to group the devices in the API service. While it is possible to do this, there are no guarantees that all devices belonging to a "Sensor" or an "Observed Property" are part of the same thematic group, especially when multiple parties register their devices into the service. For example, two different devices measuring the observed property "Temperature" may serve entirely different purposes: one sensor monitors temperature for meteorological analysis in a "Weather Monitoring" context,

while another measures parking space temperature for "Parking Lot Monitoring" applications. This same limitation applies to the "Sensor" entity, where devices grouped under a single sensor designation may span multiple thematic domains despite sharing technical specifications.

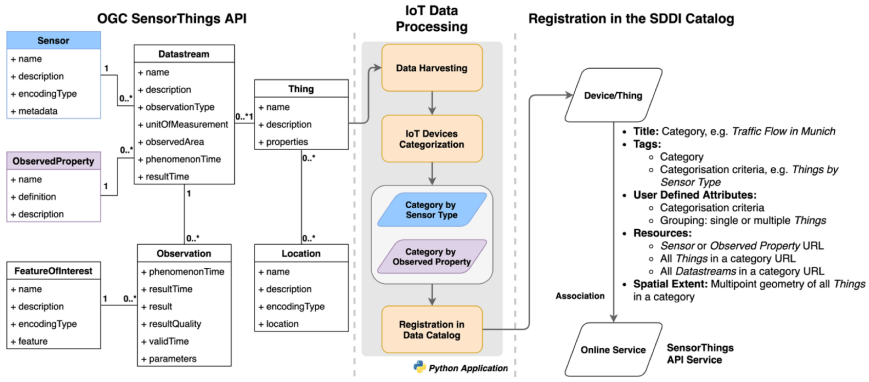

Figure 3: Workflow for automated categorization of IoT devices based on SensorThings API data model and registration in the SDDI catalog schema [3].

FIWARE's Smart Data Models provide standardized schemas for IoT data in different domains. Still, many of them represent specific objects in a particular domain, such as the "BicycleParkingStation" data model and "Streetlighting" in the mobility domain [40] [41]. While these data models help ensure interoperability between IoT devices and applications, they do not provide a comprehensive framework for classifying and registering sensor data in urban metadata catalogs. Additionally, evaluation of their practical implementation faces significant limitations due to the scarcity of publicly accessible endpoints. Although multiple cities, including Valencia, Santander, and Málaga, have adopted FIWARE-based platforms, they are not publicly available and thus cannot be used in this thesis [42].

This literature review shows that existing solutions address individual components of the IoT data management pipeline. Still, none provide a comprehensive framework specifically designed for a fully automated sensor device classification and registration in urban metadata catalogs, highlighting the need for a framework to resolve this issue.

3 Methodology

3.1 Approach

Based on the literature review, the gaps in existing IoT data integration and metadata cataloging solutions, as outlined in section 2, will be used as the basis for the design of the WRENCH (Workflow for Registration and ENriCHment of Sensor Data) framework, which this thesis will develop to automate the registration and classification of IoT sensor data into urban metadata catalogs.

The iterative development of the framework is guided by three core principles: modularity and extensibility to support future diverse IoT data sources and metadata catalogs, and automation to minimize user manual intervention. Rapid prototypes will be created and continuously tested against real-world data sources.

Clustering algorithms are evaluated based on resource requirements, accuracy, performance, and ability to a adapt to new incoming data. Due to continuous harvesting, newly collected data may not accurately reflect existing cluster representations, often requiring a re-clustering the whole dataset. Alternative approaches will be prototyped when initial solutions fail to meet satisfactory results.

The device groups created by the clustering algorithm are registered as individual entries, connected with a relationship to the API service to which these device groups belong, ensuring that these device groups are easily found within the metadata catalog while maintaining the reference to the source server.

Catalog entry titles and descriptions for the clustered device groups will be generated by a large language model (LLM) using each group's device information and metadata to create coherent, meaningful entries containing group-relevant keywords that may improve searchability of the entries within a metadata catalog.

Components are developed individually and then integrated progressively. In each integration phase, compatibility testing will be conducted with existing SDDI Catalog APIs and validation of end-to-end automation workflows.

3.2 Solution Objectives

The framework aims to achieve a scheduled, fully automated, and metadata-enriched registration pipeline. The primary objective is to automate the registration process of IoT sensor services, minimizing and potentially eliminating all manual labor required. The framework must be able to automatically extract comprehensive metadata from SensorThings API endpoints, including spatial and temporal coverage, device types, and measured parameters.

Scheduling pipeline runs with the framework will be possible, with the framework detecting added, changed, and removed devices between runs. Consequently, entries in the catalog will stay up to date with this continuous harvesting approach.

The framework's device clustering algorithm should automatically identify and group related sensors thematically. Normalized Mutual Information (NMI) is a measure of similarity between two clusters or community structures [43]. It is a normalized version of Mutual Information, which quantifies the amount of information one variable contains about another. Higher NMI values mean that predicted clusters correlate better with actual clusters. While there is no universally accepted NMI threshold, studies usually report scores of above 0.9 as indicative of strong agreement with ground truth. In network community benchmarking, top-performing algorithms often achieve $NMI \geq 0.9$ on networks with well-separated communities [44]. Hence, a minimum NMI score of 0.9 is adopted as the evaluation benchmark for the clustering performance.

In addition to structural accuracy, generated cluster descriptions must be semantically rich and incorporate diverse terminologies, improving relevance and discoverability for different user groups. The automated metadata generation process should produce catalog entries that are comparable in quality to manually created entries, as measured by completeness of required metadata fields and accuracy of descriptive content.

3.3 Evaluation Framework

The WRENCH framework is evaluated using multiple assessment methods to ensure its effectiveness.

Functional evaluation assesses the framework's automation quality, including successful data harvesting from SensorThings API

endpoints, accurate metadata attributes, and proper integration with SDDI Catalog.

Performance evaluation measures the improvements achieved through automation. Key metrics include processing duration, quality of automatically generated metadata compared to manual entries, and completeness of metadata fields. The framework's scalability is evaluated by comparing the results with data sources of different complexities and sizes.

Since accurate labels are available from the source data, clustering quality is evaluated with standard metrics that require ground truth labels. These include: Normalized Mutual Information (NMI), Homogeneity, Completeness, and V-Measure [45].

Qualitative evaluation focuses on the semantics and discoverability of generated metadata. This includes the quality and representativeness of the LLM-generated titles and descriptions.

3.4 Data Sources

The evaluation uses publicly available FROST Servers, which provide diverse IoT data from various urban domains. The primary data sources include Hamburg FROST Server [8], the Osnabrück FROST Server [6] and the Munich CUT FROST Server [7], which provides data from various domains, such as traffic management, soil and air quality, weather monitoring, and much more.

These sources represent different urban contexts and sensor types, which enables better testing of the framework's clustering algorithms across diverse thematic groupings. The Hamburg server contains diverse mobility and transportation data, while the Osnabrück server has sensors belonging to different urban domains, providing the heterogeneity needed to validate the framework's clustering capabilities. The Munich server contains mainly mobility and environment data, and represents the server with the most structured data, as it is only maintained by a single party.

The SDDI Catalog serves as the target metadata catalog for registration testing. It provides the necessary functionality for testing automated registration workflows while validating the framework's ability to generate enriched catalog entries that leverage SDDI's enhanced features, such as temporal and spatial filtering capabilities.

3.5 Limitations

The lack of diverse, publicly available FROST server datasets hinders the validation of clustering algorithms across the full spectrum of potential IoT deployment scenarios. Most FROST servers contain sensors and devices for a specific thematic domain, for example, air quality monitoring [46], water quality monitoring [47], and many others. While these endpoints still allow for testing and ensuring the general registration pipeline works, they do not allow for functional testing of the clustering algorithm.

The clustering algorithm has to be relatively fast and require minimal training. This aligns with the requirement that the registration must be close to or even fully automated, and be more efficient than manual registration. Additionally, training a neural network from scratch or fine-tuning an existing neural network is not plausible for the extent of this thesis due to time and resource constraints.

The usage of LLMs introduces a specific limitation in the context of text generation. LLMs have a limited context window. While certain proprietary LLMs such as Gemini 2.5 Flash and Pro boast a huge context window of 1 million tokens [48], their performance typically deteriorates as the number of tokens increases [49, 50]. Additionally, the scope of this thesis only includes open-source LLMs, which have shorter context windows compared to the proprietary LLMs [51]. Feeding the whole array of device information in a device group, which may consist of thousands of devices, to the LLM is not ideal for generating descriptions and titles for catalog entries. Therefore, a representative sample has to be chosen to fully depict the different devices belonging to the clustered groups.

4 WRENCH Framework

The WRENCH Framework automates the task of registering sensor services into a metadata catalog while leveraging its advantages in terms of metadata discoverability. It is a toolkit for building automated continuous data processing pipelines customized to process sensor metadata.

The framework follows a general data flow as depicted in illustration 4. The `Harvester` retrieves data from an IoT data source, such as a SensorThings API endpoint, a weather data provider endpoint, or any other sources, and creates a collection of all devices. These devices follow two paths: they are passed directly to the `MetadataEnricher` and sent to the `Grouper` for thematic clustering.

The `Grouper` runs a thematic clustering algorithm to group similar devices upon receiving the devices from the Harvester. These groups are then forwarded to the `MetadataEnricher`, which processes both the entire device collection received from the `Harvester` and the clustered groups from the `Grouper` to calculate both the service-level metadata and group-level metadata. At the last step, the `Cataloger` receives both metadata types and sends the results to the metadata catalog.

This architecture generates service-wide and granular group-specific metadata, enriching the metadata catalog with various data organization and discovery levels.

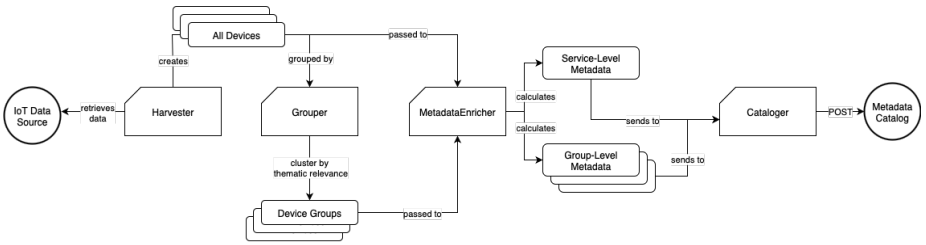

Figure 4: General flow of data in the WRENCH Framework

The framework's requirements encompass data processing pipelines with the integration of large language models and custom clustering algorithms. To keep the framework suitable for data processing purposes with good LLM integration and easy to use, the WRENCH framework will be developed in the Python 3 programming language

and installable as a package via the default `pip` installer.

WRENCH leverages Pydantic v2 for strong type checking and validation, allowing for the definition of standardized inputs and outputs for pipeline components and unified data models for interface components [52].

4.1 Data Model

To align and facilitate extensible interfaces, the framework needs to define data models to ensure compatibility between components within the framework. These data models act as the bridge between components to ensure their compatibility.

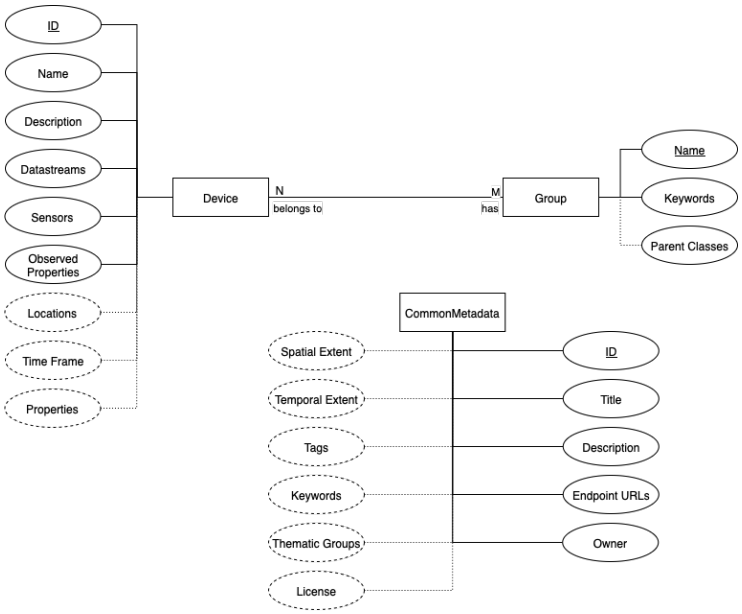

Figure 5: Entity Relationship Diagram for Data Model Representation

The few data models established in the framework represent the framework conceptually. The framework defines three core data models: `Device`, `Group`, and `CommonMetadata`. A `Device` represents a "Thing" instance which may contain any number of physical or virtual sensors. A `Group` represents a collection of devices clustered according to specific criteria—this thesis focuses solely on thematic clustering approaches. A device can belong to multiple groups if it thematically fits into them. Lastly, `CommonMetadata` stores metadata information for device collections at two scopes: service-level metadata encompassing the entire service, or group-level metadata

specific to individual groups.

Currently, the SensorThings API data model heavily influences the `Device` data model, a widely adopted standard for preparing IoT data and the framework's primary harvester implementation. Similarly, the `CommonMetadata` data model takes inspiration from the SDDI catalog data models. Since it is based on CKAN, a widespread implementation of a metadata catalog, the design aligns with established metadata catalog conventions.

As the number of harvester and cataloger implementations in the WRENCH framework grows, minor changes to the `Device` and `CommonMetadata` are expected to cater to new IoT data sources and metadata catalogs.

4.2 Base Classes

The WRENCH framework provides several abstract base classes. These abstract base classes serve as a contract for implementations, enabling different implementations to be easily interchangeable. There are four main abstract base classes: `BaseHarvester`, `BaseGrouper`, `BaseMetadataEnricher`, and `BaseCataloger`.

4.2.1 BaseHarvester

All implementations of harvesters for various IoT data sources must inherit from the `BaseHarvester` abstract base class. This abstract class defines a `return_devices` method, which returns a list of `Device` objects.

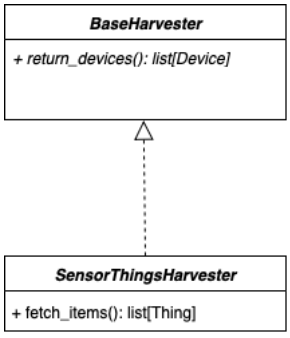

Figure 6: UML Diagram of `BaseHarvester`

WRENCH provides a ready-made `SensorThingsHarvester` which

retrieves devices from the SensorThings API based services. To retrieve it, the harvester must be supplied with a base URL representing the IoT data source.

Since the goal of the framework is to register device metadata into the metadata catalog, the harvester focuses on `Thing` entities as the primary representation of IoT devices, expanded with their `Locations`, `Datastreams`, `Sensors`, and `ObservedProperties` to capture complete device metadata. The time frames of data delivery from devices are available by extracting the outer bounds of `phenomenonTime` from the `Thing`'s `Datastreams`. The `phenomenonTime` refers to the time when the observed phenomenon occurred; however, in resource-constrained sensing devices without a clock, the empty `phenomenonTime` omitted by the client must be substituted by the current server time [2].

4.2.2 BaseGrouper

Any grouper implementation must inherit from the `BaseGrouper` abstract class, which defines a `group_devices` method that transforms device collection into clustered devices inside `Group` objects. Implementations may use any clustering algorithm if it implements this method.

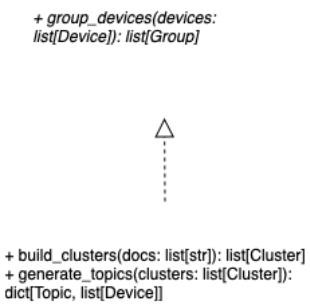

Figure 7: UML Diagram of `BaseGrouper`

The framework comes with a default grouper called the KINETIC (Keyword-Informed, Network Enhanced, Topical Intelligence Clustering) Grouper. An unsupervised network clustering technique leverages sentence embeddings and word co-occurrence network partitioning to cluster devices into groups. Details about this algorithm will be introduced in section 5.

4.2.3 BaseMetadataEnricher

The `BaseMetadataEnricher` abstracts metadata generation for service-level and group-level contexts. Since the generation process for both types of metadata is similar, this unified component handles both outputs from different pipeline stages. The enricher must correspond to an associated harvester to leverage data source-specific API knowledge for proper metadata construction. Child classes must implement the following abstract methods:

- `_build_service_urls`
- `_build_group_urls`
- `_calculate_service_spatial_extent`
- `_calculate_group_spatial_extent`

These are called by the base methods `build_service_metadata` and `build_group_metadata` to build metadata. Spatial extents require separate abstract methods because calculations may differ between service-level and group-level metadata. Both build methods must return `CommonMetadata` to be ingested by the Cataloger. The instantiation of a `BaseMetadataEnricher` requires a title and description string to be used for the service-level entry in the final metadata catalog.

The framework provides a SensorThingsMetadataEnricher to complement the SensorThingsHarvester. This component implements query URL construction for the clustered groups using filtering capabilities defined in the SensorThings API standard to filter for only devices by their ID [2]. When the number of a group's devices exceeds 100, the URLs are partitioned into multiple resource URLs to limit resource URL length and maintain browser compatibility. Consequently, more than one resource URL can be created for a single group entry if it contains more than 100 devices.

Beyond resource URLs, metadata such as spatial and temporal extents must be calculated. Temporal extent is derived from the outer bounds of all device time frames within the device collection, representing the oldest and the newest data streams across all devices in the entry. Historical devices are identified by their latest observation timestamp – when a device stops sending observations, its last recorded timestamp remains static while other devices continue updating their timestamps. The MetadataEnricher uses a threshold

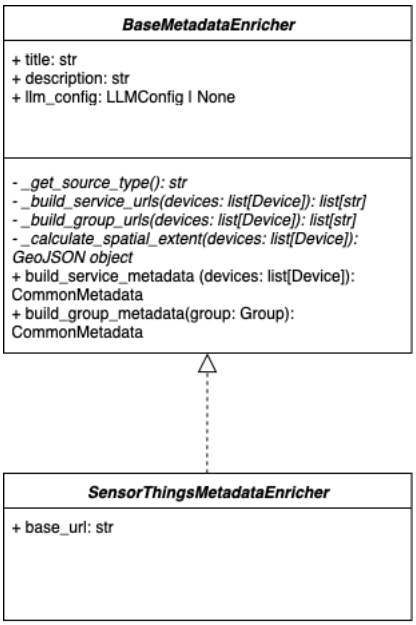

Figure 8: UML Diagram of BaseMetadataEnricher

of one day to determine if a device is still active. An active device group can be identified in the metadata catalog as an entry with a start collection date, but no end collection date. This means that devices are still sending data as of today.

Spatial information about devices in a SensorThings API service can be derived from each Thing's `Location`. Spatial extent must be represented in GeoJSON format and may be computed in several ways. For service-level entries having a vast number of devices with varying spatial features, using a bounding box is more suitable as it avoids map clutter. Group-level entries contain fewer devices and less heterogeneous spatial features; therefore, displaying devices as individual feature collections on the map benefits from increased detail and less clutter. The differences between the two spatial representations are visually displayed in figures 9 and 10.

Additionally, the title and description for group entries can be generated with the help of LLMs. When a valid LLM configuration is passed, the enricher sends metadata and sample documents to the LLM to create meaningful and informative titles and descriptions for each group entry. Representative sample devices are selected by choosing devices with unique datastream descriptions, since similar devices share the same datastream characteristics. The configuration specifies the base URL, API key (for proprietary ser-

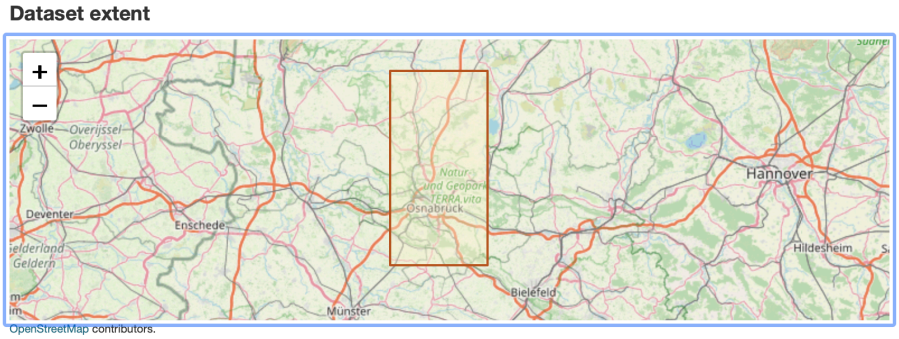

Figure 9: Example of Bounding Box Spatial Representation

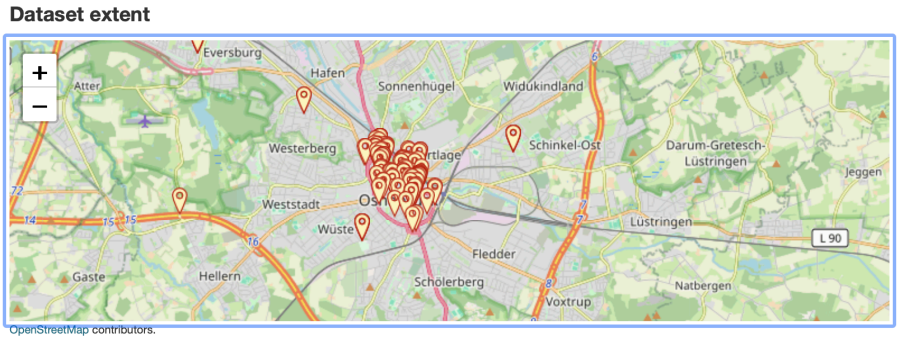

Figure 10: Example of Feature Collection Spatial Representation

vices), and model name. The system prompts and user prompts are documented in appendices B.1 for the system prompt and B.2 for the user prompt. These prompts guide the LLM in generating relevant and concise texts for entry titles and descriptions, including source server details and information extracted from sample data.

4.2.4 BaseCataloger

`BaseCataloger` is the final component of the pipeline and communicates directly with the backend metadata catalog. It exposes an abstract method `register` which must be implemented. The `register` function accepts a service-level metadata and a list of group-level metadata and creates individual entries for each service-level and group-level metadata.

The framework includes an SDDICataloger to assist in registering harvested metadata into a standardized urban metadata catalog. It

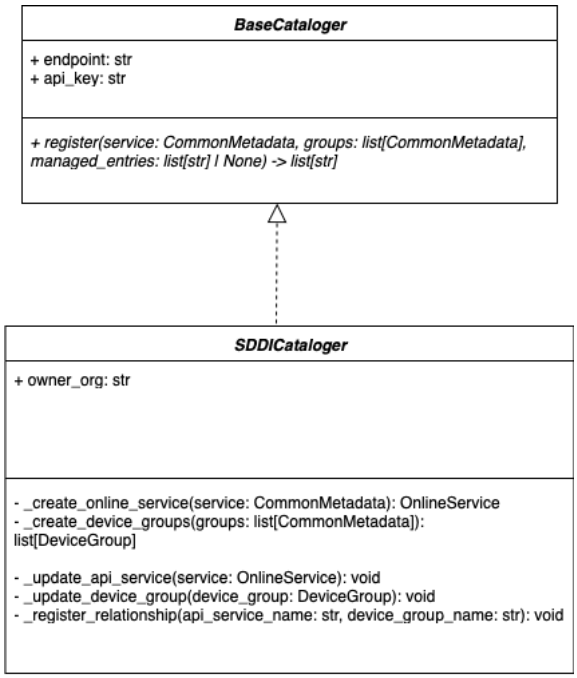

Figure 11: UML Diagram of `BaseCataloger`

leverages the SDDI catalog's features and data model definitions to enhance the discoverability of the harvested and grouped IoT source metadata. To initialize an SDDICataloger, provide the base URL, API key, and owner organization of the dataset to be registered. Every entry must have an owner assigned to it.

The Cataloger communicates with the backend catalog through an SDDI-extended CKAN API, which allows additional useful features such as geographical and temporal filtering of catalog entries. It converts the `CommonMetadata` from the enrichers into catalog-specific data models such as OnlineService and DeviceGroup. If an entry with the same ID exists in the catalog, it will try to update it with metadata from the latest run.

Each group entry and its service entry are connected via a relationship in the SDDI catalog, allowing users to visualize all a service's groups. Additionally, the temporal and spatial information calculated in the enricher can be used inside the metadata catalog to filter for entries, spatially or temporally. The Cataloger also leverages groups defined in the SDDI project reference. Service-level entries get assigned to the main group "Online Service", and device group entries get assigned to "Device/Thing". Using the clustering

results from the Grouper, the entries can also be assigned to one of the 15 thematic groups defined in the SDDI catalog reference [20].

4.3 Pipeline Mechanism

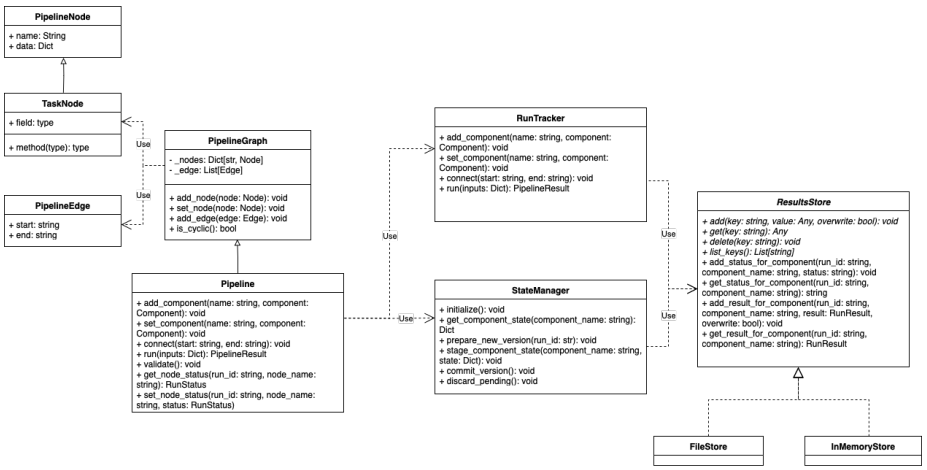

Figure 12: Overall pipeline architecture of the WRENCH framework, including its dependencies

The pipeline mechanism is inspired by the Neo4j GraphRAG Package for Python [53]. The original mechanism is simplified and modified to meet the needs of building a data processing pipeline for urban IoT devices. Instead of having components for Graph Retrieval Augmented Generation (GraphRAG), the WRENCH framework has components such as harvesters, groupers, and catalogers, which, when assembled, create an automated registration pipeline. It keeps various useful features from the original implementation, such as YAML file-based pipeline configuration, component connection validation, and preconfigured template pipelines. Additionally, WRENCH includes several extension features to enable pipeline scheduling and state management. Figure 12 depicts the resulting pipeline architecture overview.

A pipeline is created by connecting nodes with edges. Nodes are an abstraction over pipeline components, while the edges connect nodes. Edges are validated to prevent incompatible components from being linked, ensuring that input and output types at each connection point correspond.

Pipeline execution follows component connection paths, with each

node returning a `RunResult` containing output data and a `RunStatus` indicating execution status.

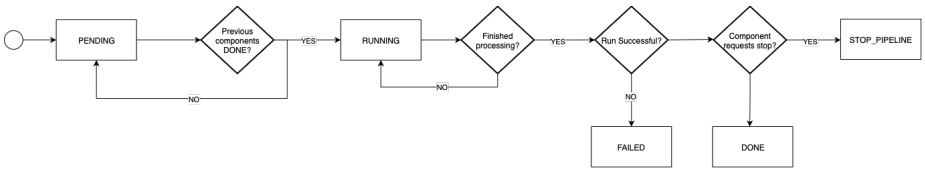

Figure 13: State transition diagram for a node's RunStatus

`RunStatus` can be one of 5 values: `PENDING`, `RUNNING`, `DONE`, `STOP_PIPELINE`, and `FAILED`. Starting a pipeline sets every node's `RunStatus` into the PENDING state. PENDING nodes only transition to RUNNING once all the previous nodes finish processing or reach the DONE status. RUNNING nodes imply that the components are still running and processing data. Once the component completes its task, it transitions into either:

- FAILED, meaning the component run resulted in an error.
- DONE, meaning the component run successfully finished, or,
- STOP_PIPELINE, signaling that pipeline execution doesn't need to continue.

Moreover, the pipeline maintains a `PipelineRunStatus` reflecting its overall execution state. The `PipelineRunStatus` can be one of "STARTED", "COMPLETED", "FAILED", or "STOPPED". Each pipeline run, including its run_id and final `PipelineRunStatus`, is stored in a `RunRecord`. The `PipelineRunTracker` manages these records for observability and debugging purposes.

Although the internals of the pipeline mechanism are relatively complex and convoluted, the framework provides preconfigured template pipelines that make it easy for users to get started. Users must only provide source data and target catalog information to start running registration pipelines. The rest of the work is handled internally by the framework.

4.3.1 Components

Components are building blocks of a registration pipeline. They must implement a correctly typed `run` method, which returns a

subclass of `DataModel`, the base class for all component outputs and allows for the pipeline to define standard attributes for a typical component output, also enabling the pipeline to easily dump the data into JSON format using Pydantic `BaseModel` methods.

The pipeline uses `run` method to execute its components, and the type annotations allow the pipeline to validate if the output of a node matches the expected input of a subsequent node.

Components can be stateful or stateless. In a continuously running pipeline (i.e., scheduled pipeline), a stateful component looks up previous states and compares the latest state with the results of the current execution. Depending on what the component does, it may stop pipeline execution early if no state changes are found, or execute using only the changed items. The behavior of a stateless component does not change whether it runs in a scheduled or a non-scheduled manner.

A registration pipeline consists of the following essential components: Harvester, Grouper, MetadataEnricher, and Cataloger. These components are a wrapper around the abstract base classes introduced in 4.2 and make them usable in the pipeline.

Harvester

The harvester component retrieves data from IoT data sources and converts these source-specific data formats into the framework's standardized data model. It wraps the `BaseHarvester` implementations and handles state change detection. The harvester returns `Items`, which consist of devices and operations.

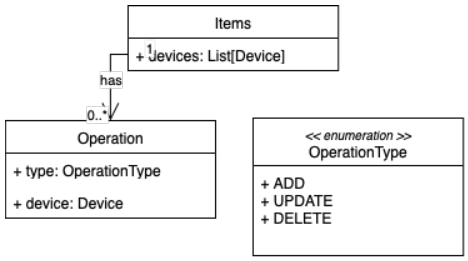

Figure 14: `Items` and `Operation` Data Model represented in a UML Diagram

`Operation` represents the change needed to transform the previous state into the current state. Each `Operation` corresponds to a device and tracks modifications made to the device from the last

state. There are three types of operations: ADD, UPDATE, and DELETE. Add operations handle new devices appearing in current state. Update operations mark modified existing device metadata that has changed. Delete operations represent removed devices that don't exist anymore in the source.

Scheduled pipelines with stateful harvesters manage their state throughout pipeline runs and discover which devices have been added, removed, or changed. These changes are translated into operations and propagated to the following components.

Grouper

The grouper component puts devices into logical groups and manages incremental structure changes. It manages group state transitions during pipeline executions and wraps around `BaseGrouper` implementations. The Grouper takes devices and operations as input (output of a harvester) and returns `Groups` containing groups (a collection of `Group`). It inherits from the `DataModel` class and thus can be used as the output of the Grouper.

Figure 15: `Groups` Data Model represented in a UML Diagram

The component holds group state across runs in scheduled pipelines with stateful groupers. It maintains existing groups incrementally. When provided with operations, the Grouper executes them by type (add/update/delete) and tracks which groups have been modified. These restricted changes allow the pipeline to only propagate modified groups to other components, which is more useful in large-scale deployments.

When the Grouper has a previous state and no operations are received, the Grouper can also signal to terminate remaining pipeline processes early, leading to avoided work for processing unchanged data. When no previous state exists or for initial runs, the Grouper processes all devices and sets up a complete group structure.

MetadataEnricher

The metadata enricher component wraps the `BaseMetadataEnricher` implementations, which enriches device and group metadata for their catalog registration. The metadata builder takes a collection of `Device`, `Operation`, and `Group` as input and returns `Metadata`, which consists of `service_metadata` and `group_metadata`.

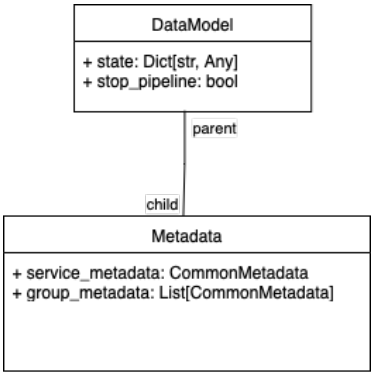

Figure 16: `Metadata` Data Model represented in a UML Diagram

For service-level metadata, the component rebuilds service metadata in every run to reflect the latest state of all devices, ensuring that the entries stay up-to-date. Group metadata is handled differently since the component employs LLM to generate metadata titles and descriptions. The WRENCH framework uses the OpenAI Python SDK to send requests to various LLM backends. For testing the framework, this thesis uses an internally provided Ollama server, meaning any LLMs available in Ollama can be used to generate topics.

Since LLM generation is not deterministic, these titles and descriptions must only be generated once and preserved on the first run in the component state. Subsequent runs can then use this information stored in the state so that regeneration is not required. If the contents of the source deviate too much and a re-clustering of the devices is triggered, the MetadataEnricher generates new entries for these new groups and removes old groups.

In scheduled executions, the metadata builder maintains previous group metadata state and selectively updates only the groups affected by operations. This approach balances metadata freshness with consistency, ensuring established group descriptions remain stable while accommodating changes.

Cataloger

The cataloger component registers generated Metadata into metadata catalogs and manages the lifecycle of catalog entries. It wraps the `BaseCataloger` implementations and handles the final registration step of the pipeline. The cataloger takes `service_metadata` and `group_metadata` as input and returns `CatalogerStatus`, which indicates a success status and contains lists of registered groups.

The component maintains the state of previous catalog registrations to properly manage catalog entries across pipeline runs, including updating existing entries, creating new ones, and cleaning up no longer valid entries. The cataloger handles the complexity of different catalog APIs and registration protocols through its wrapped `BaseCataloger` implementation.

4.3.2 Scheduling and State Management

WRENCH enables scheduling of pipeline executions. Users can configure the schedule using two options: `IntervalScheduler` and `CronScheduler`. With `IntervalScheduler`, users specify a duration after which every pipeline must rerun. `CronScheduler`, on the other hand, accepts a cron expression and reruns pipelines as frequently as defined with the expression. The scheduler is based on the Advanced Python Sheduler (APScheduler) library. To schedule pipelines, the pipeline instance must be wrapped inside a scheduler.

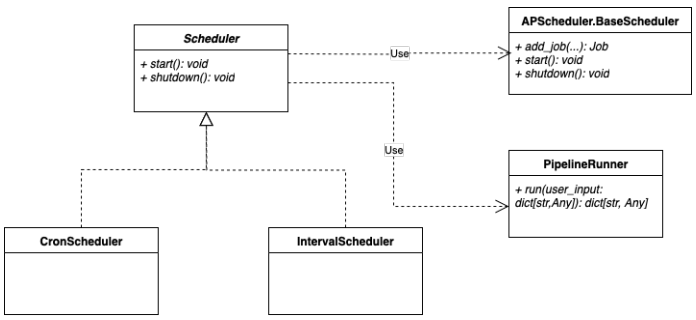

Figure 17: WRENCH Scheduler UML Diagram

Pipelines have a `PipelineStateManager` which tracks component states and versions. The manager initializes states for new components or uses the latest existing state for comparison with current states. Component states are committed only upon successful pipeline completion or early termination. Failed pipeline runs

result in complete state rollback. This transaction-like mechanism prevents state inconsistencies between different components.

4.3.3 State Persistence

The framework implements persistent state management through the `ResultStore` abstraction to ensure robust operation and traceability. Without persistence, pipeline execution history and component states can be lost during system outages, requiring a complete rerun of compute-intensive processes.

The framework provides two implementations of `ResultStore`:

- `FileStore` persists pipeline states as JSON files inside a store directory.
- `InMemoryStore` stores pipeline states in memory.

`FileStore` provides fault tolerance by keeping execution state in the file system, keeping the data available through system interruptions. `InMemoryStore` offers faster performance for development scenarios where execution speed is prioritized over persistence.

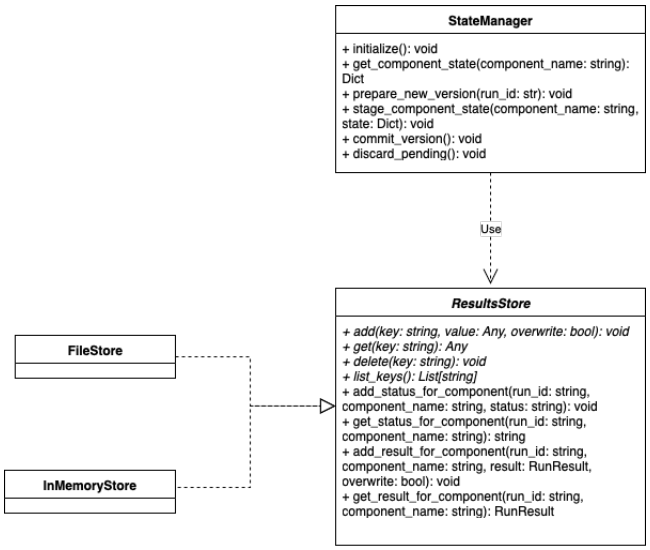

Figure 18: Pipeline state persistence implementation and usages

Both `PipelineRunTracker` and `PipelineStateManager` depend on an implementation of `ResultStore`, making it flexible for users to choose which store implementations to use.

4.3.4 Template Pipelines

Template pipelines are pre-built pipeline structures with predefined components and connections. They allow users to run registration pipelines without manually configuring pipeline topology. They offer the fastest way to get started since they provide an abstraction over the complexity of component orchestration. Without understanding the entire pipeline architecture, users can implement the required interfaces, and the template handles connecting the components and running them in the correct order automatically.

The framework includes a preconfigured sensor registration template pipeline that implements the workflow depicted in Figure 4. The template requires an implementation of the following interfaces: `BaseHarvester`, `BaseGrouper`, `BaseMetadataEnricher`, and `BaseCataloger`, and optionally a scheduler configuration. If no scheduler configuration is given, the pipeline will just run once.

Template pipelines support two instantiation approaches: direct Python instantiation and YAML configuration. Direct Python instantiation involves creating ainstance `SensorRegistrationPipeline` instance and passing in the above dependencies.

Program 1 Direct Python instantiation of `SensorRegistrationPipeline` example

```
  pipeline = SensorRegistrationPipeline(
    harvester=my_harvester,
    grouper=my_grouper,
    metadatabuilder=my_metadata_builder,
    cataloger=my_cataloger,
    scheduler_config=optional_scheduler
)
```

With YAML configuration, users must create a YAML file with specified component implementations and configuration. An example of this configuration file can be found in the appendix A. The `PipelineRunner`'s `from_config_file` class method accepts the YAML configuration file, parses through it, nd generates the pipeline definition accordingly. The scheduler must be created separately since configurations are parsed by `PipelineRunner` and the scheduler requires a `PipelineRunner` as input.

Program 2 Instantiation of a pipeline through a YAML configuration file

```
pipeline_runner = PipelineRunner.from_config_file(
    "./pipeline_config.yaml"
)

config = IntervalSchedulerConfig(interval="PT10M")

scheduler = config.create_scheduler(pipeline_runner)

try:
    scheduler.start()
    await asyncio.Event().wait()
except (KeyboardInterrupt, SystemExit):
    scheduler.shutdown()
```

5 The Clustering Algorithm

As outlined in section 1.3, part of the registration process is to create feasible device groups representing the contents of the data source to improve their visibility and discoverability in the metadata catalog. This framework's Grouper component achieves this by clustering the devices according to their thematic groups. The thematic groups will be closely following the 15 groups defined in the SDDI catalog: Administration, Urban Planning, Environment, Health, Energy, Information Technology, Tourism, Living, Education, Construction, Culture, Trade, Mobility, Craft, Work, and Agriculture [20].

Cities collecting IoT data typically aim to publish sensor data to enable the development of Urban Digital Twins. The sensors available in these services will likely be related to one of the topics mentioned in the list above. It is also important to note that this IoT data may come from physical devices or simulations.

Classification approaches may be helpful in cases where pre-defined sensor classification schemas are available and aligned. While many data models are available to represent different groups in different domains, it is not within the scope of this thesis to align all these data models and define a granular classification schema encompassing all the domains mentioned above. A topic modeling approach eliminates the need to define a generalized classification schema.

Based on the literature review, various models can be used to group sensor devices. These include the conventional LDA (Latent Dirichlet Allocation) [32], newer NTM (Neural Topic Modeling)

models like BERTopic [35], or even graph-based topic modeling like CWUTM (Topic model based on co-occurrence word networks for unbalanced short text datasets) [36]. Section 6 will compare and evaluate these different models.

5.1 The Source Data

The evaluation data used comes from the Hamburg, Osnabrück, and Munich FROST server. These servers are used as part of the attempt to enable the urban digital twin and the smart city. Hence, the IoT data in these platforms is heterogeneous and spans multiple domains. The data are primarily in German with some English content. Since the data within these servers continuously changes and evolves, the following exploration reflects the server state at the time of writing.

Hamburg FROST Server

The Hamburg FROST server data can be accessed through https://iot.hamburg.de/v1.1 and contains 4953 Things as of the time of writing this thesis. The devices in the server mainly come from the mobility domain. These include: vehicle traffic counters, bicycle traffic counters, simulated road network monitoring, electric vehicle charging station availability, and city-bike station availability. It also has a single Thing from the energy domain measuring the production and usage of renewable energy infrastructure in the Hamburg University of Applied Sciences: Energie-Campus.

The distribution of the Things by device type is displayed in the pie chart in Figure 19. Road network simulation takes the most significant slice of the pie with 2226 Things. As can be observed, the distribution of devices by category is skewed, with categories like "Road Network Monitoring" having more than 2000 entries, while other essential categories like "Renewable Energy Infrastructure" only have a single entry. The final clustering algorithm chosen must be able to identify these small device clusters within the server to adequately and appropriately represent the contents of these servers in the metadata catalog.

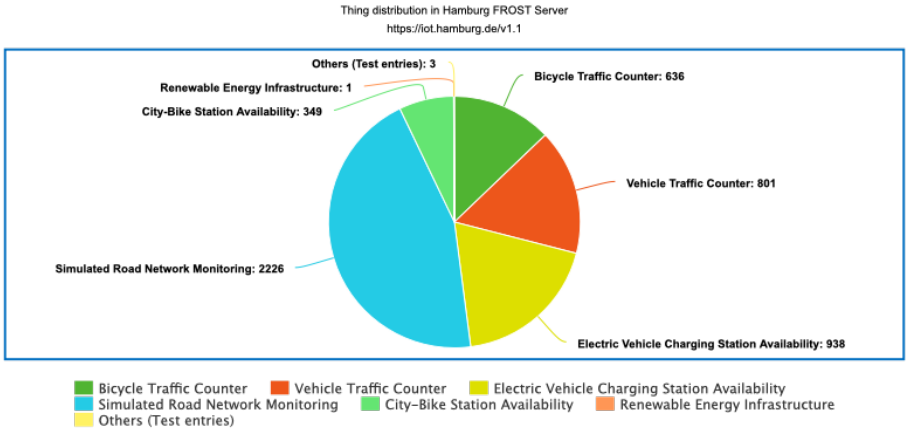

Figure 19: Thing distribution in Hamburg FROST server by category

Osnabrück FROST Server

The Osnabrück FROST server is available at the URL https://daten-api.osnabrueck.de/v1.1 and has 478 Things. The server is more heterogeneous than the Hamburg FROST server and spans the energy, environment, and mobility domains.

The mobility domain has devices monitoring parking lot availability, traffic counts, and electric vehicle charging station availability. The environment domain is represented by groundwater level monitoring, soil monitoring, and weather monitoring devices, with one device monitoring "Tiny House" indoor living conditions and another monitoring road conditions for winter service. It also has some historical records of a 2023 cycling event monitoring the number of cyclists passing a track, and a daily drinking water consumption monitor.

The distribution of Things in the Osnabrück FROST server is even more fragmented than the Hamburg FROST server. It contains individual categories with a single Thing, such as "Indoor Living Monitoring" and "Winter Service Sensor", while also having categories with a high number of Things, like "Groundwater Level Monitoring" and "Traffic Counter".

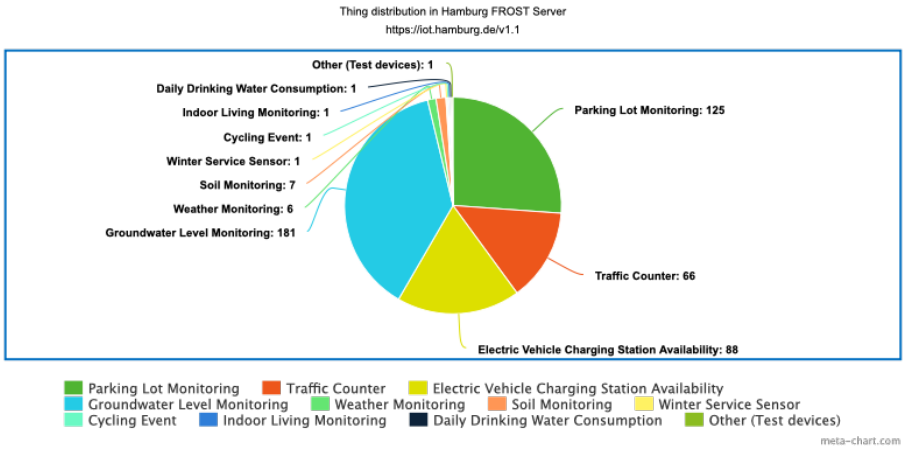

Figure 20: Thing distribution in Osnabrück FROST server by category

Munich FROST Server

The Munich FROST server is accessible through the URL https://cut.gis.lrg.tum.de/frost/v1.1 and has 1397 Things as of 23 July 2025. The server primarily focuses on the mobility and environment domain, featuring diverse sensor devices including air quality monitoring sensors, traffic light signal status, traffic flow monitoring sensors. Since the Chair of Geoinformatics at TUM maintains the server, the data is better organized and more consistently structured than other data sources that rely on multiple contributors or less centralized management.

The Figure 21 shows the distribution of sensors by category in the Munich FROST server. The server has fewer sensor categories compared to other FROST servers observed in this thesis. However, like the Hamburg and Osnabrück servers, Munich also shows an uneven distribution of categories, with some containing significantly fewer devices than others.

5.2 Clustering Requirements

The clustering requirements derive from the source data. The data exploration shows that the number of devices for a single device category can be highly skewed; some categories may include thousands of devices, while others may have fewer than 10 devices. The clustering method must be capable of identifying different clusters

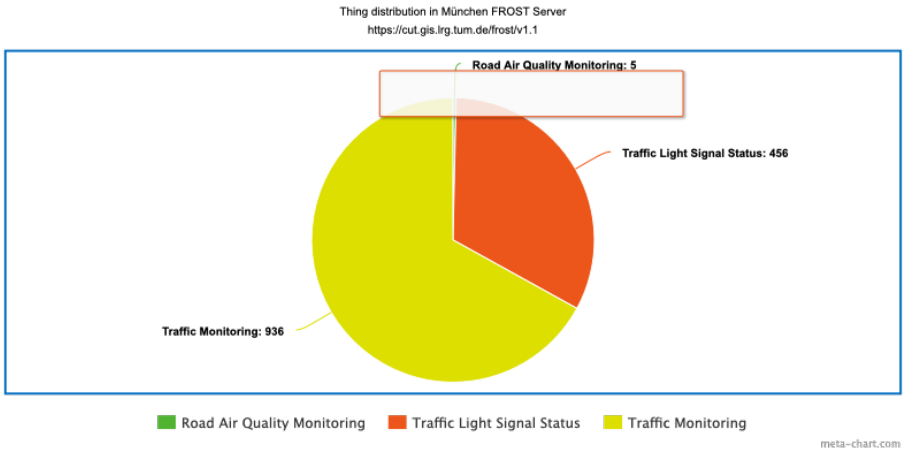

Figure 21: Thing distribution in Munich FROST server by category

of varying sizes.

Data varies across different IoT servers. The source data introduced in the previous subsection maintains data in various domains. The Hamburg FROST server focuses more on mobility data, while the Osnabrück FROST server contains data from a more diverse set of domains. These server-specific categories are latent and must be discovered by the clustering method.

Typically, Things are registered using template texts. An example from the Hamburg FROST server in table 2 shows this. The *Zählfeld* text template is used for bicycle counting devices, and the *Stadtrad-Station* template for city-bike station availability.

Thing ID	**Name**	**Description**
8098	Zählfeld B_11.2_2_G	Zählfeld zur Zählung der von der Infrarotkamera erfassten Mobilitätswerkzeuge
8100	Zählfeld G_40.7_2_G	Zählfeld zur Zählung der von der Infrarotkamera erfassten Mobilitätswerkzeuge
9331	E-Ladestation DE*HHM*E560	Electric charging station for electric cars
84	StadtRad-Station aab32389-22bc-325a-af60-6eb525ffdc56	StadtRad-Station U Lattenkamp / Bebelallee
137	StadtRad-Station bf822969-6f7a-3bb4-b00c-fddef19fa23f	StadtRad-Station U Niendorf-Nord / Nordalbingerweg

Table 2: Sample Things from Hamburg FROST server [4]

Datastream names and descriptions across similar Things are also derived from similar templates.

This discovery provides the foundation for a keyword extraction-based semantic clustering. The hypothesis is that Things with similar functionality will derive from the same template and show lexical similarity in their names and descriptions, enabling keyword-based clustering to identify functional groups within the IoT device ecosystem.

Sensors and ObservedProperties are ignored during clustering, since they may incorrectly link unrelated clusters—for example, two applications that use CO2 sensors would be grouped despite serving different purposes. Other entities, such as Observations and Locations, are completely ignored since they don't bring semantic value to improve clustering.

Therefore, the clustering methodology utilizes only Thing and Datastream entities to capture semantic relationships. This selective approach takes advantage of the template-derived lexical similarities in device names and descriptions while avoiding the clustering interference caused by shared technical components or semantically irrelevant data, ensuring accurate identification of functionally related device groups.

5.3 KINETIC Topic Modeling

Current topic modeling methods like LDA and BERTopic suffer from an intrinsic hyperparameter trade-off, which prevents effective detection of big and small clusters simultaneously. Parameter settings optimized for main topics neglect underrepresented clusters entirely, while settings optimized for minority topics fragment large clusters into incoherent pieces. This limitation restricts their ability to provide adequate topic coverage for larger and smaller clusters. The evaluation of clustering results in section 6 highlights this problem.

Moreover, multiple studies have shown that LDA performs poorly on short texts [54–56]. Therefore, a graph-based co-occurrence network topic modeling approach is introduced here as KINETIC (Keyword Informed, Network Enhanced, Intelligent Clustering).

5.3.1 Methodology

KINETIC is a graph-based topic model that leverages keyword co-occurrence networks and community detection to partition the co-occurrence network into communities or clusters. It uses a mix of Sentence Transformer embeddings [57] and term frequency to calculate similarities between documents and clusters. These classified clusters are then passed on to an LLM, which generates appropriate names for each cluster.

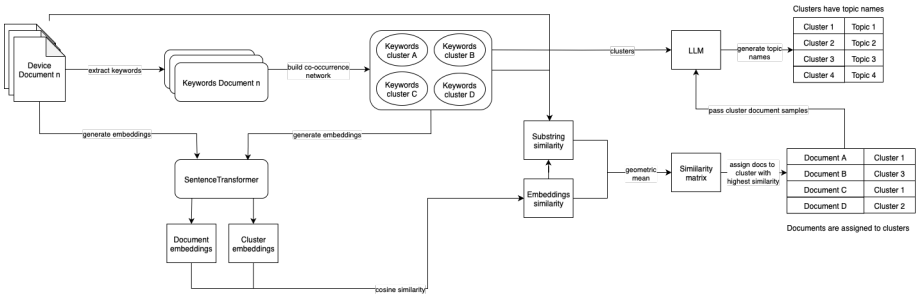

Figure 22: Full workflow of KINETIC topic modeling process

5.3.2 Keyword Extraction

The first step of KINETIC involves extracting keywords from the device documents. Device documents are stringified `Device` data models with selected fields (Name, Description, and Datastreams) as described previously in section 5.2. These documents are passed to a keyword extraction method such as KeyBERT or YAKE!.

KeyBERT is a keyword extraction technique that uses BERT embeddings to identify relevant and representative keywords most similar to a document [58]. By leveraging BERT embeddings, semantically similar keywords can be extracted by computing the cosine similarity of the word or phrase embeddings with the document embeddings. KeyBERT has many input parameters that help determine how to extract keywords. Several key parameters include:

- `keyphrase_ngram_range`: range of word number in each keyword
- `top_n`: number of top keywords to extract
- `stop_words`: words to ignore in the keyword extraction

KeyBERT offers more parameters to fine-tune keyword extraction, but those listed above are the most critical for configuration.

YAKE! (Yet Another Keyword Extractor) is a keyword extraction method which leverages statistical text features extracted from single documents to select the most relevant keywords for a text [59]. YAKE! also provides several tuning parameters similar to KeyBERT:

- `n`: number of words in each keywords
- `lan`: language for removing stop words
- `top_n`: number of top keywords to extract

Both methods offer multilingual support. KeyBERT leverages pretrained multilingual Sentence Transformer models. YAKE is language-independent since it only uses statistical features from the text document and provides a list of stop words for different languages.

The emergent keywords form the foundation for thematic clustering because they reflect the semantic meaning of each device. Keywords should appropriately capture the diverse domains in the IoT service to create meaningful clusters.

5.3.3 Keyword Clustering

After extracting keywords from each document, these are clustered to form keyword groups representing topics. The clustering method involves constructing and partitioning a keyword co-occurrence network into different communities or clusters. Keywords are represented as network nodes and their co-occurrences as weighted edges.

Two keywords co-occur when they appear in the same document. Edge weights correspond to co-occurrence count, with higher weights indicating frequent co-occurrences. This method takes advantage of device names, descriptions, and datastreams that stem from similar templates described in section 5.2.

The co-occurrence network can be partitioned using community detection methods such as the Louvain method, which is widely used to detect non-overlapping community structures of large networks using modularity optimization [60]. Furthermore, the Louvain method can be configured to consider weighted edges, utilizing the

co-occurrence frequency information to build communities within the network. The method also takes a `resolution` parameter, referring to the time described in [61], which can control the size of detected communities. Higher values favor smaller communities, while lower values produce larger communities.

The detected keyword communities represent the topics available in a sensor service. These communities provide a foundation for thematic organization but require further processing to enable device assignment. To achieve this, each community must be represented in a continuous embedding space where semantic relationships can be quantified and devices can be systematically assigned to their most relevant thematic groups.

5.3.4 Embedding Generation

Embeddings are generated for the keyword clusters and device documents to semantically classify devices into keyword clusters. They must be embedded using the same model to ensure consistent representation within the same embedding space.

The input data comes in German and English; hence, a multilingual pretrained Sentence Transformers model is required to correctly represent the data in the embedding space. There are several options for using multilingual models.

The Sentence Transformers library provides a selection of benchmarked multilingual models, such as the ”paraphrase-multilingual-MiniLM-L12-v2” and ”distiluse-base-multilingual-cased-v1” models; however, it also supports using embedding models from Hugging Face. The embedding leaderboard in Hugging Face provides a comparison of text and image embedding models across thousands of languages [5].

To keep the memory consumption and running time of the embedding generation minimal, the best-performing models with reasonably low memory consumption and smaller embedding dimensions are selected. Two open source models, ”multilingual-e5-large-instruct” and ”Qwen3-Embedding-0.6B,” are chosen for testing.

Query prompts guide the embedding generation with these models to give context for embedding documents and clusters.

Rank	Model	Mem. Usage	Size	Dim	Score
1	Qwen3-Embedding-8B	28.9 GB	7B	4096	74.0
2	Qwen3-Embedding-4B	15.3 GB	4B	2560	72.3
3	gemini-embedding-001	N/A	N/A	3072	71.8
4	Qwen3-Embedding-0.6B	2.3 GB	595M	1024	66.8
5	multilingual-e5-large-instruct	1.1 GB	560M	1024	64.9
6	text-multilingual-embedding-002	N/A	N/A	768	64.6
7	Linq-Embed-Mistral	13.6 GB	7B	4096	62.2
8	GritLM-7B	13.8 GB	7B	4096	61.8
9	gte-Qwen2-7B-instruct	29.0 GB	7B	3584	61.5
10	SFR-Embed-Mistral	13.6 GB	7B	4096	60.0

Table 3: Top 10 Embedding Models based on classification score from Hugging Face Leaderboard (State: 12 July 2025) [5]

Prompt 1: Query prompt for document embedding generation

```
Emphasize keywords in the document which might help
differentiate between different categories of urban
sensor topics:
```

Prompt 2: Query prompt for cluster embedding generation

```
In the context of urban sensors, distinguish these
individual clusters from other sensor types and
categories.
```

5.3.5 Similarity Calculation and Classification

The classification of devices into the formed keyword clusters uses a similarity calculation. This calculation considers two different similarity parameters, semantic and substring similarity.

The generated embeddings for device documents and keyword clusters are used to measure conceptual relatedness, and the similarity between these embeddings can be calculated using cosine similarity, providing a similarity score. The cosine similarity between their embedding vectors $\mathbf{d}_i, \mathbf{c}_j \in \mathbb{R}^d$ is defined as:

$$S_{semantic}(i,j) = \frac{\mathbf{d}_i \cdot \mathbf{c}_j}{\|\mathbf{d}_i\|\|\mathbf{c}_j\|} \tag{1}$$

where $\mathbf{d}_i$ represents the embedding vector for the i-th document,

and $\mathbf{c}_j$ represents the embedding vector for the j-th cluster.

Secondly, counts of cluster keyword occurrences represent substring similarity between a document and a cluster. The substring occurrences are normalized by the keyword count of each cluster to keep similarity scores comparable across different cluster sizes. The substring similarity between document i and cluster j is defined as:

$$S_{\text{substring}}(i, j) = \frac{1}{|K_j|} \sum_{w \in K_j} \text{count}(w, d_i) \tag{2}$$

where K_j represents the set of keywords for cluster j, $|K_j|$ denotes the number of keywords in cluster j, and count (w, d_i) represents the frequency of keyword w in document d_i.

The geometric mean of both substring and semantic similarity is then calculated to produce a unified similarity score. This dual approach balances semantic understanding and explicit keyword matching. The combined similarity between document i and cluster j is defined as:

$$S_{\text{combined}}(i, j) = \sqrt{\frac{S_{\text{semantic}}(i, j)^2 + S_{\text{substring}}(i, j)^2}{2}} \tag{3}$$

where $S_{\text{semantic}}(i, j)$ represents the semantic similarity from Equation 1, and $S_{\text{substring}}(i, j)$ represents the substring similarity from Equation 2.

Outlier devices are rarely discovered unless a new set of devices with a new topic is added to the source server. When this happens, the clusters must be regenerated to correctly represent the new set of devices. Outlier detection is employed using the interquartile range rule [62] to account for this. However, experiments show that the usual 25th and 75th percentiles create excessive outlier classifications; therefore, a wider range of 10th and 90th percentiles is used to reduce false positives. Predictions below the lower bound of $Q_{0.1} - 1.5 * (Q_{0.9} - Q_{0.1})$ are treated as outliers and are not classified into any of the existing clusters.

5.3.6 LLM Topic Generation

Large language models (LLMs) assist the KINETIC process by creating appropriate names for each cluster. These named clusters are

stored in the Topic data model. Passing the data model enforces the LLM to output a structured data model with expected values.

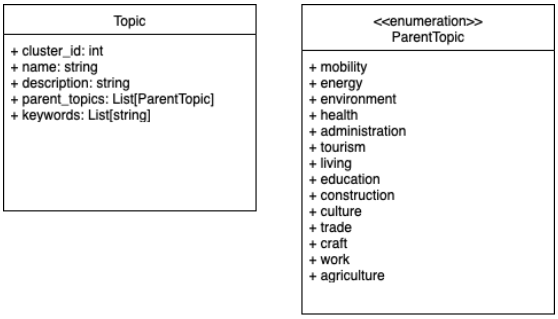

Figure 23: The Topic data model for LLM Structured Output

The Topic data model provides several essential fields. The `cluster_id` identifies which cluster this topic corresponds to, `name` and `description` are both used to name and describe the clusters. `ParentTopic` is an enumeration of the SDDI-defined thematic groups. These groups guide the creation of the topic names and descriptions, while enforcing the groups to at least be related to one of the SDDI-defined thematic groups so that they can be appropriately assigned to their thematic group when registering into the SDDI catalog [20]. The "Information Technology" category is excluded due to the LLM's tendency to consistently classify all sensor data into this category, regardless of the actual data type or domain.

The OpenAI Python SDK supports structured output by passing in a Pydantic Base Model [52]. The Topic data model inherits from it and therefore can be used as the enforced output data model for the LLM.

To generate comprehensive and representative topic names and descriptions, the cluster keywords are taken, along with some sample data that belongs to the cluster. Similar to the approach used in the `BaseMetadataEnricher` in section 4.2.3, representative sample data are selected by choosing devices with unique datastream descriptions, since similar devices share the same datastream characteristics. Since the classification of devices into clusters happens before the topic generation with LLM, these classified devices can be used as sample data. This information is injected into the user prompt and used in the generation.

The system prompt and user prompt used in this part are documented in the appendices B.3 and B.4.

The model `llama3.3:70b-instruct-q4_K_M` is used for testing as it is the best available model within the internal Ollama instance and has a context length of 128k tokens. It is based on the multilingual Llama 3.3 model with 70b parameters, pretrained and fine-tuned for following instructions with 4-bit quantization for computational efficiency [63]. The model balances speed and performance for generating name and description texts based on the system and user instructions.

This final step in the KINETIC processing flow produces the final output of the KINETIC grouper, a set of topics, with name, description, and representative keywords, along with devices classified to each topic.

5.3.7 Cache

KINETIC uses LLM to generate topic names and descriptions, making the generated texts non-deterministic by nature. This poses a problem since inconsistent naming could result in duplicate entries where the same clusters receive similar but different names across different runs.

To address this challenge, KINETIC maintains a cache system to keep clusters, their embedding representations, and generated topics from previous runs. This approach not only ensures consistency, but also reduce the running time and memory requirements for the subsequent runs since cluster embeddings are cached and won't require recalculation.

Furthermore, maintaining and loading clusters and topics from the cache makes it possible to perform manual intervention by modifying the cache. If generated topics and clusters are unsatisfactory, a human can manually review and improve these clusters before subsequent runs.

The cache is only invalidated when KINETIC detects many new outliers, meaning that a new cluster of devices may have been added to the source data. The cache is overwritten in this case, and a cluster regeneration is triggered.

6 Discussion and Evaluation

To determine how the framework performs compared to manual registration, qualitative and quantitative evaluations are performed to assess the automated registration pipeline's efficiency, performance, and quality. The assessment uses the Hamburg, Osnabrück, and Munich FROST server data described in section 5.1

First, the various grouper components must be evaluated to determine the best component for the whole pipeline. Since the KINETIC grouper consists of several building blocks, these also need to be assessed to select the best parts that make up the KINETIC grouper.

6.1 KINETIC Component Selection

The KINETIC process requires multiple components to work together to provide a full topic modeling method. Multiple potential candidates will be used in various parts of the process.

Keyword Extraction

As introduced in section 5.3.2, the keyword extraction process can employ two potential methods: KeyBERT and YAKE!.

Testing keyword extraction using both KeyBERT and YAKE! shows that using an `ngram_range` of 1 yields best results. Extracting multi-word keywords leads to a more unique set across different documents. Since the clustering algorithm relies on keyword co-occurrences across documents, identical multi-word keywords are less likely to appear in multiple documents, reducing the effectiveness of co-occurrence-based clustering.

The table 4 shows both KeyBERT and YAKE!, when configured to return multi-word keywords, extract a more unique set of keywords, e.g., "parkplatz nettebad" is not identical to "parkplatz kamp", and these are considered different instances of keywords in the co-occurrence network.

While the example in table 4 shows that YAKE! and KeyBERT return similar keywords for the same data, KeyBERT performs better at extracting semantically more meaningful keywords. Table 5 shows an example where this property is apparent. Since YAKE re-

Text	KeyBERT (1-ngram)	KeyBERT (2-ngram)	YAKE! (1-ngram)	YAKE! (2-ngram)
Parkplatz Nettebad 01 Parkplatz (Standort: Nettebad 01) {'Belegtstatus an dem Parkplatz Nettebad 01', 'Temperatur an dem Parkplatz Nettebad 01'}	nettebad, parkplatz, 01, standort, temperatur, belegtstatus	nettebad 01, parkplatz nettebad, 01 parkplatz, temperatur parkplatz, standort nettebad	nettebad, parkplatz, standort, belegtstatus, temperatur	parkplatz nettebad, nettebad, parkplatz, standort, belegtstatus
Parkplatz Kamp 02 Parkplatz (Standort: Kamp 02) {'Temperatur an dem Parkplatz Kamp 02', 'Belegtstatus an dem Parkplatz Kamp 02'}	kamp, parkplatz, standort, 02, temperatur, belegtstatus	02 parkplatz, kamp 02, parkplatz kamp, temperatur parkplatz, parkplatz standort	kamp, parkplatz, standort, belegtstatus, temperatur	parkplatz kamp, kamp, parkplatz, standort, belegtstatus, temperatur

Table 4: Keyword Extraction Comparison: KeyBERT vs YAKE! Methods with 1-ngram and 2-ngram

lies on statistical features of the text document, words that appear often, such as location details, are of higher importance. KeyBERT extraction prioritizes semantic value and can be guided with a query prompt.

Embeddings Generation

Since the KINETIC process uses sentence embeddings heavily, the model selection for embedding generation is a critical decision and needs to be appropriately evaluated. Section 5.3.4 explores the available model choices and provides two options: "multilingual-e5-large-instruct" and the "Qwen3-Embedding-0.6B" models.

Unfortunately, the machine used for testing does not have adequate memory to run the "Qwen3-Embedding-0.6B" model smoothly in a reasonable amount of time. Therefore, the "multilingual-e5-large-instruct" model is selected as it yields good results while keeping the memory and compute requirements low.

Text	KeyBERT	YAKE!
GWM Hansalinie Grundwasserpegelmessung (Standort: Hansalinie) {'Umgebungstemperatur an der GWM Hansalinie', 'Wasserpegel an der GWM Hansalinie', 'Differenzdruck an der GWM Hansalinie', 'Atmosphärischer Druck an der GWM Hansalinie', '(Wasser-)Temperatur an Sonde der GWM Hansalinie'}	grundwasserpegel-messung, wasserpegel, umgebungstemper-atur, gwm, differenzdruck, wasser, temperatur	hansalinie, gwm, standort, wasser, temperatur, wasserpegel, differenzdruck
GWM Vördener Weg Grundwasserpegelmessung (Standort: Vördener Weg) {'Umgebungstemperatur an der GWM Vördener Weg', '(Wasser-)Temperatur an Sonde der GWM Vördener Weg', 'Wasserpegel an der GWM Vördener Weg', 'Atmosphärischer Druck an der GWM Vördener Weg', 'Differenzdruck an der GWM Vördener Weg'}	grundwasserpegel-messung, wasserpegel, umgebungstemper-atur, gwm, wasser, standort, temperatur	weg, vörderner, gwm, standort, wasser, wasserpegel, differenzdruck

Table 5: Keyword Extraction Comparison for Groundwater Monitoring Stations

6.2 Topic Modeling Methods

Section 5.3 introduced several alternative clustering methods to KINETIC, such as LDA and BERTopic. This section evaluates these methods and explains how KINETIC performs better than conventional methods. The clustering methods will be assessed based on the clusters created, the accuracy of the devices clustered, and how well the generated LLM entries represent the cluster.

Cluster quality is evaluated by comparing the clusters generated by each method using the device groups assessed in Figure 19, 20, and 21. The generated clusters should reflect the pie chart distribution as closely as possible. The device assignments to clusters are also evaluated to see how well the methods group devices together. Since ground truth labels are available, assessment of each clustering method uses metrics requiring true labels [45].

6.3 Cluster Quality

Latent Dirichlet Allocation (LDA)

The LDA method requires the initialization of several parameters. To ensure the usage of the best parameters, a grid search guided by the perplexity [32] is run to find the best hyperparameters to run the topic modeling. Using this method, the following hyperparameters are determined:

- For Hamburg; nr_topics: 5, alpha: 0.01, beta: 0.1
- For Osnabrück; nr_topics: 10, alpha: 0.01, beta: 0.1
- For Munich; nr_topics: 5, alpha: 0.01, beta: 1

The LDA method generates the following clusters shown in table 6 for Hamburg, table 7 for Osnabrück, and 8 for Munich.

Group Keywords	**Devices**	**Inferred Category**
Road, Segment, Class	2,226	Simulated Road Network Monitoring
Stadtrad, Station, Der	205	City Bike Station Availability
Station, Stadtrad, Der	147	City Bike Station Availability
Hhm, Charge, Ladepunkt	941	Electric Vehicle Charging Station Availability
Hlstelle, Verkehrsz, Im	1,436	Vehicle Traffic Counter

Table 6: LDA Grouper Results for Hamburg Dataset

In all datasets, LDA performs well in identifying major clusters such as ”Simulated Road Network Monitoring” and ”Electric Vehicle Charging Station Availability” in the Hamburg dataset, as well as ”Groundwater Level Monitoring” and ”Parking Lot Monitoring” in Osnabrück dataset, and ”Traffic Flow Monitoring” in and ”Traffic Light Signal Status” in Munich dataset.

The device counts for each category closely match the ground truth data. For example, the LDA topic model identified 2226 devices for ”Simulated Road Network Monitoring”, matching the ground truth count, while other categories show similar accuracy in device assignments. The topic model achieves near-perfect clustering results for the Munich dataset, only marred by a single misclassification where the ”Traffic Flow Monitoring” cluster was incorrectly split into two separate groups.

Topic ID	Top Keywords	Devices	Inferred Category
Topic 0	Mrwash, Stra, Enring	10	Unclear
Topic 1	Der, Busse, Personen	47	Unclear
Topic 2	Dem, Bodensensor, Cm	16	Soil Monitoring
Topic 3	Parkhaus, Aktuelle, Auslastung	24	Parking Lot Monitoring
Topic 4	Gwm, Der, Atmosph	122	Groundwater Level Monitoring
Topic 5	Ladepunkt, Ladestation, Elektrofahrzeuge	88	Electric Vehicle Charging Station Availability
Topic 6	Gwm, Der, Alfhausen	59	Groundwater Level Monitoring
Topic 7	Der, Strup, Personen	10	Unclear
Topic 8	Nettebad, Kollegienwall, Amtsgericht	0	Unclear
Topic 9	Parkplatz, Temperatur, Dem	102	Parking Lot Monitoring

Table 7: LDA Topic Modeling Results for Osnabrück Dataset [6]

On the other hand, the Osnabrück and Hamburg datasets illustrate the weakness of LDA for minority device categories. This issue is particularly pronounced in Osnabrück, where the diverse sensor environment has numerous minority clusters that LDA is unable to identify. The algorithm generates spurious and redundant clusters with irrelevant keywords rather than identifying meaningful device groupings. The dominant sensor categories are not identified, e.g., "Weather Monitoring" in Osnabrück and "Renewable Energy Infrastructure" in Hamburg, highlighting the weakness of LDA for imbalanced datasets.

The dramatic difference in performance in Munich from the rest of the datasets demonstrates LDA's sensitivity to data structure and distribution. Munich's achievement stems from its well-defined sensor types and similarly sized clusters, which enable LDA to detect even minor minority clusters like "Air Quality Monitoring" (5 devices) as well as large clusters with hundreds of devices. The generated topic keywords in Munich are semantically dense and descriptive, thereby proving that LDA is as efficient as any other clustering algorithm when provided with appropriately structured data.

LDA emerges as the fastest method to group the devices, requiring only 26 seconds for the Hamburg dataset, 3 seconds for

Group Keywords	Devices	Inferred Category
lsa, signalgeber, status, signal, lights, head, vehicle, flow, loop, lfu	376	Traffic Light Signal Status
flow, passenger, vehicle, schleifen, lorries, type, detektor, speed, cars, inductive	936	Traffic Flow Monitoring
vehicle, matter, lorries, loop, lights, lfu, inductive, index, hours, hourly	80	Traffic Flow Monitoring
lfu, dioxide, average, sensor, hourly, monoxide, particulate, aqi, carbon	5	Air Quality Monitoring

Table 8: LDA Grouper Results for Munich Dataset [7]

the Osnabrück dataset, and finally 8 seconds to cluster the Munich dataset.

BERTopic

BERTopic uses both Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN) and Uniform Manifold Approximation and Projection (UMAP) as part of its process and requires initialization parameters for these components to be set [33, 34]. HDBSCAN is an extended version of DBSCAN that converts it into a hierarchical clustering algorithm. UMAP is a dimensionality reduction technique that maps the high-dimensional embeddings into a lower-dimensional space while keeping the features of the high-dimensional space intact.

To ensure the generalizability of the findings, extensive hyperparameter optimization is avoided, since it may lead to overfitting to specific dataset characteristics and reduce the transferability of results to similar corpora. However, several standard parameters such as `min_topic_size` from HDBSCAN and `n_neighbors` from UMAP need to be tuned to get the algorithm to work.

The BERTopic runs for Hamburg, Osnabrück and Munich datasets mostly use the sensible default parameters, with some adjustments to the `min_topic_size` parameter. Since the clusters in the Hamburg dataset are expected to be bigger, a higher `min_topic_size` of 150 is set. For the Osnabrück dataset, a smaller value of 10 is chosen for its smaller cluster sizes. Using both 10 and 150 for the Munich dataset yields the same results.

Topic ID	Devices	Keywords	Inferred Category
Topic_0	2,226	road, segment, flow, class, min, the, simulation, network, monitored, model	Simulated Road Network Monitoring
Topic_1	1,187	hlstelle, verkehrsz, intervall, im, an, kfz, der, aufkommen, infrarotdetektor	Vehicle Traffic Counting
Topic_2	921	charge, hhm, de, electric, to, availability, point, according, ocpp, ladepunkt	Electric Vehicle Charging Station Availability
Topic_3	348	stadtrad, station, der, fahrr, gbarer, verf, an, einer, belegungssensor, lastenr	City Bike Station Availability

Table 9: BERTopic Clustering Results for Hamburg Dataset

Experiments with BERTopic for both the Hamburg, Osnabrück, and Munich datasets result in the topics displayed in tables 9, 10, and 11 respectively.

Based on the observations, BERTopic performs comparably better than LDA, presenting coherent and non-overlapping topics for the Hamburg dataset. However, it still has trouble detecting minority clusters for both datasets.

BERTopic discovers multiple duplicated entries for "Parking Lot Monitoring" and "Groundwater Level Monitoring" for the Osnabrück dataset. Still, it fails to find clusters with fewer devices, such as "Soil Monitoring", "Weather Monitoring", or "Tiny House / Indoor Living Monitoring".

Table 11 shows the results of the Munich dataset. BERTopic performs worse than LDA here, failing to detect minority cluster "Air Quality Monitoring", misclassifying several "Traffic Flow Monitoring" devices, and creating multiple clusters for "Traffic Light Signal Status". The selected keywords for the clusters are also unrepresentative, containing multiple stop words and uninformative words such as "for" and "any".

BERTopic is much slower than LDA, requiring 617 seconds for Hamburg, 152 seconds for Munich, and 96 seconds for Osnabrück. Despite this significant computational cost, BERTopic does not yield significant improvements in clustering quality over LDA. In most instances, BERTopic is worse, forming less cohesive clusters and missing important device categories that were identified by LDA.

Topic ID	Devices	Keywords	Inferred Category
Topic_0	89	gwm, der, an, atmosph, rischer, druck, wasserpegel, differenzdruck	Groundwater Level Monitoring (General)
Topic_1	84	ladepunkt, ladestation, ocpp, according, availability, elektrofahrzeuge	Electric Vehicle Charging Station Availability
Topic_2	75	parkplatz, dem, parksensor, placepod, pni, zustand, belegtstatus	Parking Lot Monitoring
Topic_3	31	alfhausen, gwm, der, im, an, rechtiener, ranlage, eversburg	Groundwater Level Monitoring (Alfhausen)
Topic_4	17	busse, personen, hlung, anzahl, lkws, motorraeder, fahrraeder, pkws	Vehicle Traffic Counting
Topic_5	25	wittenfelde, gwm, teufelsheide, der, an, pelkebach, differenzdruck	Groundwater Level Monitoring (Wittenfelde)
Topic_6	20	thiener, alfhausen, dorf, damm, gwm, esch, der, an	Groundwater Level Monitoring
Topic_7	19	parkplatz, moskaubad, schinkelbad, nettebad, parking, lot, bosch	Parking Lots Monitoring
Topic_8	15	parkhaus, auslastung, aktuelle, garage, parkhauses, eines, osnabr	Parking Lot Monitoring
Topic_9	14	schlagvorderstrasse, parkplatz, cke, nr, hasebr, dem, hs	Parking Lot Monitoring
Topic_10	12	wasserwerk, thiene, zum, gwm, der, an, differenzdruck, wasserpegel	Groundwater Level Monitoring (Wasserwerk)

Table 10: BERTopic Clustering Results for Osnabrück Dataset

KINETIC

Unlike other methods introduced in this topic, KINETIC provides very few hyperparameters to run the clustering. KINETIC only needs to know what language the dataset is in and which embedding model needs to be used. Both of these have default values that are easily determined and don't need fine-tuning.

Another parameter `resolution` controls the size of the clusters created. Additionally, an LLMConfig is needed to generate the cluster name and descriptions.

With these simple requirements, KINETIC produces clusters displayed in table 12 for the Hamburg dataset and table 13 for the Osnabrück dataset.

For the Hamburg dataset, KINETIC performs excellently at identifying all the clusters, including the minority clusters such as "Renewable Energy Infrastructure" and even detects less relevant test

Topic ID	Devices	Keywords	Inferred Category
Topic_0	914	for, traffic, passenger, flow, cars, any, speed, detektor, schleifen	Traffic Flow Monitoring
Topic_1	324	signalgeber, lsa, status, signal, lights, head, traffic	Traffic Light Signal Status
Topic_2	80	ta, signalgeber, lsa, status, signal, lights, head, traffic	Traffic Light Signal Status
Topic_3	52	signalgeber, lsa, status, signal, lights, head, traffic	Traffic Light Signal Status
Topic_4	27	for, traffic, flow, passenger, any, of detektor, cars, schleifen, speed	Traffic Flow Monitoring

Table 11: BERTopic Clustering Results for Munich Dataset

entries. It fails to detect or differentiate between "Bicycle Traffic Counter" and "Vehicle Traffic Counter" and groups them in the same cluster; it misses the vehicle-related keywords and only finds the bicycle-related keywords.

After passing the clusters to the LLM, the LLM identified the irrelevant clusters `cluster_1` and `cluster_3` and generated topics for the rest of the clusters.

Osnabrück dataset's clustering results demonstrate KINETIC's ability to identify semantically meaningful groupings from the diverse sensor landscape. As shown in Table 13, it successfully differentiated between various monitoring domains, from environmental sensors like weather stations and groundwater monitoring to urban infrastructure such as parking lots and electric vehicle charging stations. Notably, KINETIC was able to discover several specialized clusters that previous methods failed to detect, including "Indoor Living Monitoring" and "Cycling Event."

However, the quality of the generated clusters varies significantly based on the semantic richness of the original keywords. Clusters with highly specific and contextually relevant terms, such as "Weather Monitoring" (containing windrichtung, luftfeuchtigkeit, windgeschwindigkeit) and "Groundwater Level Monitoring" (containing grundwasserpegelmessung, wasserpegel, differenzdruck), provide clear indicators that enable accurate LLM-based topic generation. In contrast, clusters with more generic terms, location-specific identifiers, or device codes provide limited semantic information, making accurate topic description more challenging.

The Osnabrück dataset's fragmented nature presents additional

Cluster ID	Keywords	Inferred Category
cluster_0	intervall, mobilitätswerkzeuge, verkehrszählstelle, zählung, woche, infrarotkamera, fahrradaufkommen	Bicycle Traffic Counter
cluster_1	dataintegrations, datastream, integrations, update, data, test, thing	Others (Test entries)
cluster_2	station, ladestation, ladepunkt, electric, charging, de, 01	Electric Vehicle Charging Station Availability
cluster_3	infrastructure, mqtt, http, health, check	Others (Test entries)
cluster_4	stadtrad, fahrräder, lastenräder, straße, platz, bahnhof, chaussee	City-Bike Station Availability
cluster_5	segment, road, monitored, 20, network, flow, 10	Simulated Road Network Monitoring
cluster_6	methanisierungsanlage, photovoltaik, verbrauch, energie, hausanschluss, elektroautoladestation, energies	Renewable Energy Infrastructure

Table 12: KINETIC Clustering Results for Hamburg Dataset [8]

complexities for KINETIC. While the algorithm successfully identified all major sensor categories, it missed some minority clusters, such as the "Winter Service Sensor" and "Daily Drinking Water Consumption" groups.

The Munich dataset presents a special challenge to topic generation by KINETIC, particularly for "Traffic Light Signal" devices. These sensors contain minimal contextual information, cryptic device names, short descriptions, and sparse metadata on datastream. Consequently, KINETIC's keyword extraction retrieves mainly device abbreviations and IDs rather than semantically useful terms. While this limitation does not reduce the clustering algorithm's ability to group similar devices, it significantly hampers from the LLM's ability to generate coherent topic descriptions. This can be seen in Table 14, where `cluster_2` presents the resulting incoherent keyword patterns.

The KINETIC cache system provides a solution to this problem. By storing cluster information in a `clusters.json` file that is persisted across runs, users can manually edit keyword representations for problematic clusters by modifying the cache file directly. For the Munich dataset, the "Traffic Light Signal" cluster keywords can be manually refined to include more semantically meaningful terms,

Cluster ID	Keywords	Inferred Catagory
cluster_0	standort, parkplatz, belegtstatus, temperatur, behinderung, 01, 02	Parking Lot Monitoring
cluster_1	ladestation, elektrofahrzeuge, ladepunkt, osnabrück, parkhaus, 04, auslastung	Electric Vehicle Charging Station Availability
cluster_2	grundwasserpegelmessung, wasserpegel, gwm, umgebungstemperatur, wasser, differenzdruck, alfhausen	Groundwater Level Monitoring
cluster_3	verkehrszählung, busse, verkehrsteilnehmer, pkws, straße, rosenplatz, lkws	Traffic Counter
cluster_4	bodensensor, bodenfeuchte, bodentemperatur, baum, pflanzenverfügbares, tiefe, ema	Soil Monitoring
cluster_5	schiebetür, geräte, klimaanlage, schränke, zukunft, stromverbrauch, ausgestattet	Indoor Living Monitoring
cluster_6	wetterstation, windrichtung, luftfeuchtigkeit, wetter, windgeschwindigkeit, lufttemperatur, luftdruck	Weather Monitoring
cluster_7	ergebnisse, fahrten, fahrt, verkehrsmenge, stadtradeln, dauer, strecke	Cycling Event

Table 13: KINETIC Clustering Results for Osnabrück Dataset [6]

enabling the LLM to generate appropriate topic descriptions in subsequent runs. For example, the number keywords in the ”Traffic Light Signal Status” clusters can be replaced with the keywords ”ampel”, ”light”, ”signal”, resulting in a more meaningful keyword array.

While the degree of automation is reduced in these scenarios, it provides flexibiltiy for users when defining their clusters and its representative keywords.

KINETIC clustered the Hamburg dataset in 1613 seconds, the Osnabrück dataset in 211 seconds and the Munich dataset in 1045 seconds. These runtimes are considerably longer than previously introduced methods, but still reasonable to run clustering methods. Moreover, with the cache enabled, subsequent runs of the Hamburg dataset took only 630 seconds, considerably less than the initial running time and comparable to BERTopic runs. The most time-consuming parts of KINETIC are the keyword extraction and document embedding process. The cached run stores clustered keywords

Cluster ID	Keywords	Inferred Catagory
cluster_0	so2, pm10, ozone, o3, monoxide, lfu, deby115	Air Quality Monitoring
cluster_1	lorries, traffic, speed, detektor, rate, any, of	Traffic Flow Monitoring
cluster_2	signalgeber, lsa, 13, 484, 480, 426, 76	Traffic Light Signal Status

Table 14: KINETIC Clustering Results for Munich Dataset (Before manual adjustments) [7]

and doesn't need to run keyword extraction, therefore cutting down a significant amount of time from running the grouper for a second time.

6.4 Classification Accuracy

Classification accuracy is evaluated using the metrics Normalized Mutual Information (NMI) score, Homogeneity, Completeness, and V-Measure.

A clustering result is homogeneous when all clusters contain only data points from a single class. If there are classes without data points, homogeneity decreases [64].

Completeness evaluates if all the data points of a given class are elements of the same cluster. Lower scores mean the clustering algorithm fails to put data points of the same class together. Often, the homogeneity and completeness of a clustering method are inversely related. A naive clustering method, which clusters every data point into one cluster, has perfect completeness, since all data points belong to the same cluster; however, homogeneity is extremely low as other classes are ignored [64].

V-Measure is an entropy-based measure introduced by the authors Rosenberg and Hirschberg [64] as a measure of how successfully the completeness and homogeneity criteria are fulfilled. It is calculated as the harmonic mean of completeness and homogeneity, similar to how F-measure combines precision and recall scores [65].

The resulting scores for evaluation of classification performance in table 15 show that KINETIC performs best for both Hamburg and Osnabrück datasets, achieving near-perfect V-Measure results for the Hamburg dataset and a minor decline in accuracy for the Osnabrück dataset.

Method	NMI	Homogeneity	Completeness	V-Measure
LDA Hamburg	0.999	0.998	0.999	0.999
LDA Osnabrück	0.820	0.939	0.727	0.820
LDA Munich	0.896	1.000	0.812	0.896
BERTopic Hamburg	0.907	0.971	0.850	0.907
BERTopic Osnabrück	0.674	0.854	0.556	0.674
BERTopic Munich	0.774	0.966	0.645	0.774
KINETIC Hamburg	**0.999**	**0.999**	**1.0**	**0.999**
KINETIC Osnabrück	**0.940**	**0.934**	**0.946**	**0.940**
KINETIC Munich	**1.0**	**1.0**	**1.0**	**1.0**

Table 15: Classification Performance Scores for Topic Modeling Methods

LDA method also performs nearly on par with KINETIC for the Hamburg and Munich datasets. However, its performance declines by a noticeable margin on the Osnabrück dataset. This suggests that topic diversity and cluster imbalance have a heavier impact on classical topic modeling methods such as LDA. The Osnabrück dataset, in particular, contains more minority clusters, which are more challenging to detect.

BERTopic performs the worst out of the three explored methods, although it follows a similar performance trend across the Hamburg and Osnabrück datasets. Its performance declines even more sharply than LDA on the Osnabrück dataset.

The three methods present a trade-off between topic modeling accuracy and computational efficiency. KINETIC offers higher accuracy at the cost of increased runtime. In contrast, LDA offers faster processing but is more susceptible to higher performance degradation when applied to an imbalanced dataset or one containing minority clusters.

6.5 WRENCH Framework Performance

6.5.1 Processing Efficiency

Processing time measurements are conducted on an Apple Silicon M1 processor and an 8 GB RAM system. The following table shows each component's processing times and memory consumption for both source datasets. To reduce the effects of system-level noise, e.g., background processes consuming CPU or memory resources, memory consumption, and processing times are averaged over three runs.

The components used in evaluating the framework performance are: SensorThingsHarvester (Harvester), KINETIC (Grouper), SensorThingsMetadataEnricher (MetadataEnricher), and SDDICataloger (Cataloger).

Component	Dataset	Peak memory	Proc. time
Harvester	Hamburg (4,953 Things)	514 MB	86.2 s
	Osnabrück (478 Things)	473 MB	11.9 s
	Munich (1397 Things)	494 MB	30.5 s
Grouper	Hamburg	850 MB	1631.8 (630.6) s
	Osnabrück	468 MB	211.7 (191.4) s
	Munich	513 MB	991 (234.6) s
MetadataEnricher	Hamburg	812 MB	59.6 s
	Osnabrück	493 MB	89.2 s
	Munich	527 MB	23.7 s
Cataloger	Hamburg	727 MB	19.8 s
	Osnabrück	488 MB	22.1 s
	Munich	497 MB	5.4 s
Total Pipeline	Hamburg	917 MB	1813.4 s
	Osnabrück	534 MB	337.3 s
	Munich	595 MB	1121.3 s

Table 16: Component Performance by Dataset

Table 16 presents the runtime and the peak memory usage of each component in the framework. As expected, the KINETIC grouper is the most compute-intensive component, since it performs several compute-intensive calculations for embeddings generation, similarity calculation, and keyword extraction. These tasks are typically more efficient when executed on a GPU, which the test machine does not have, resulting in subpar performance. The Hamburg dataset's processing time is 1813.4 seconds (approximately 30 minutes). However, using the KINETIC cache can significantly improve this, reducing the processing time by over 1000 seconds (around 16 minutes).

Figure 24 illustrates how each framework component's processing time grows with increasing dataset size. The Grouper scales the most with higher dataset size, with processing time increasing exponentially from 211.7 to 1631.8 seconds for Osnabrück and Hamburg datasets respectively. The Harvester shows more linear scaling, increasing from 11.9 to 86.2 seconds across the same range. This performance difference reflects the complexity of each component's operations. While the Harvester performs straightforward data collection and the MetadataEnricher processes aggregated metadata, the Grouper must analyze semantic relationships between thousands of individual device descriptions, making it the primary bottleneck in the pipeline. The Total Pipeline performance closely follows the

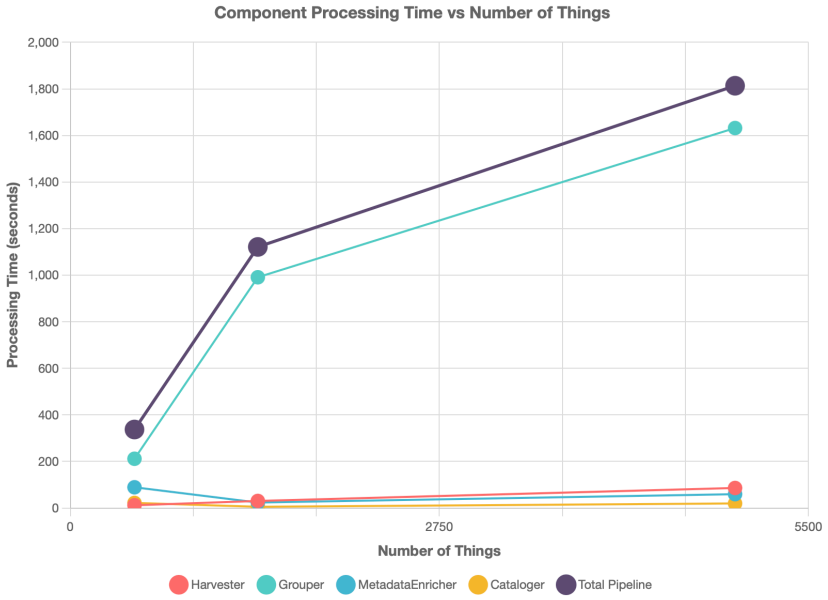

Figure 24: Component Processing Time Comparison to Dataset Size

Grouper's behavior, confirming that clustering operations dominate the overall processing time.

6.5.2 Quality of LLM-Generated Entries

The quality of the final device group entries generated by the LLMs must also be evaluated for completeness, coherence, and relevance. Both the generated description and title must answer the following questions:

- What are the devices in this group measuring?
- To which server does this group belong?
- How many devices are in this group?
- What kind of sensor is used by these devices?
- Does the description reflect all the different devices in the group?

Using KINETIC, the group names and descriptions generated are

shown in appendices C.1 for the Hamburg FROST server,C.2 for the Osnabrück FROST server, and C.3 for the Munich FROST server.

The titles and descriptions generated for the Hamburg dataset demonstrate good quality and structure. While the "Road Network Monitoring" entry's description correctly identifies the data as simulated, including this information in the title would improve clarity and visibility. The "Bicycle Sharing and Availability" entry describes specific bike types. Since the grouper could not successfully differentiate between bicycle and vehicle counts for "Traffic Counting and Monitoring", the description includes this information. The LLM successfully generated and gave information about the counting targets (bicycles and vehicles). It also provided details about data aggregation intervals and sensor types.

The Osnabrück device groups are also well represented by the device group entries in the cluster. The entry description for "Weather Station Data and Atmospheric Conditions" is exceptionally well crafted since it includes relevant keywords such as "wind speed and direction", "rainfall", "UV index", and "temperature", which enhances searchability and discoverability within the metadata catalog. The "Electric Vehicle Charging Infrastructure" entry contains misclassified "Parking Status" devices; these are accurately reflected in the entry description as "current occupancy of parking facilities". While not ideal, this reflects the upstream clustering limitations rather than deficiencies in the LLM's text generation capabilities.

The pipeline run for Munich dataset results in three group-level entries being created. The "Traffic Flow Analysis" entry exhibits clear entry title and informative descriptions, containing information about the inductive loop detectors used for collecting traffic flow data and the recorded parameters such as traffic speed, flow rate, and concentration. Moreover the "Air Quality Monitoring" entry is also very well described with details about each measured pollutants and the time interval of the provided data.

All the entries exhibit information about the sensor types used in these device groups, the number of devices in each entry, and the properties the devices measure. The entry titles include the source server identification to avoid confusion and increase visibility. However, the generated descriptions lack publisher and ownership information of the dataset, which represents a limitation for users requiring contact details for dataset inquiries or collaboration purposes.

6.6 Catalog Results

The pipeline execution generates catalog entries within an SDDI catalog test instance. These catalog entries utilize built-in SDDI catalog features. Entries require a main category and optionally a thematic category assigned to it. The thematic categories, although optional, aid in discoverability by grouping entries into a predefined set of themes. Every entry the pipeline generates is assigned to at least one of these thematic categories by providing a tag with the category name when registering the entry. Tags can be seen under the "Data and Resources" section. Figure 25 shows the tag "mobility" for the group entry "Bicycle Sharing and Availability from Hamburg FROST Server".

The MetadataEnricher component uses an LLM to generate both the title and description during the metadata enrichment step. The SDDI catalog's search functionality uses the generated content, improving discovery of entries. For example, users searching for "e-cargo bikes," "stadtrad," or "bike-sharing" will get this catalog entry in their search results, even if these exact terms only appear in the entry description.

The "Data and Resources" section directly links to the original data source. Since the test instances use SensorThings API services, these URLs point to filtered endpoints that display only Things relevant to each catalog entry. Multiple URLs are provided when necessary to prevent filter expressions from exceeding length limits.

Below the dataset extent, which provides spatial information on where the devices in this entry are located, are additional metadata information such as author name, email, and temporal information. Figure 26 shows how this looks in the catalog. The framework calculates the start and end of the validity period for the dataset and the GeoJSON expression, resulting in the dataset's extent view on the embedded map. If the dataset's end of validity period is within a day, the dataset is considered as still delivering data, and the end of validity period field will be empty. These temporal and spatial information enable users to filter entries by providing a time frame or a geographical boundary.

The user must provide other information, such as author, maintainer, language, and version, as part of the metadata since those cannot be discovered from the underlying dataset.

Relationships can be visualized between group-level and service-

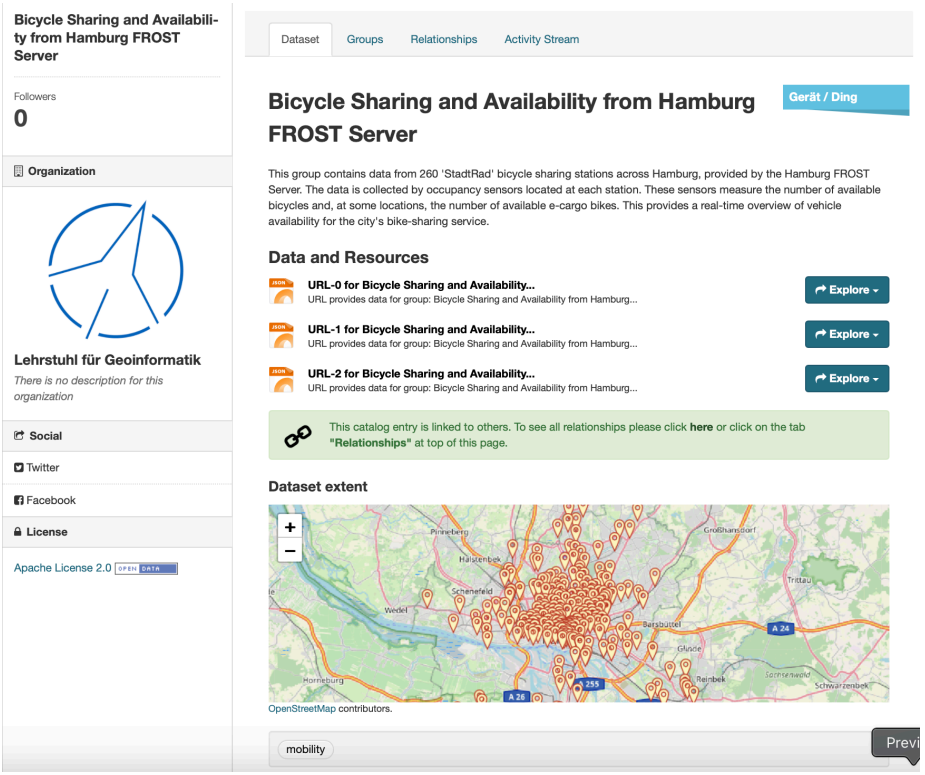

Figure 25: Generated Group Entry in SDDI Catalog from Pipeline Execution

level entries in the ”Relationship” tab as seen in Figure 27. This relationship tab enables users to navigate between device groups and source servers easily. The relationship can be viewed in both ways. Service-level entries such as the Hamburg FROST servers have many nodes or entries pointing to it, whereas typically a group-level entry such as ”Bicycle Sharing” only has a single arrow pointing to the service-level entry it belongs to.

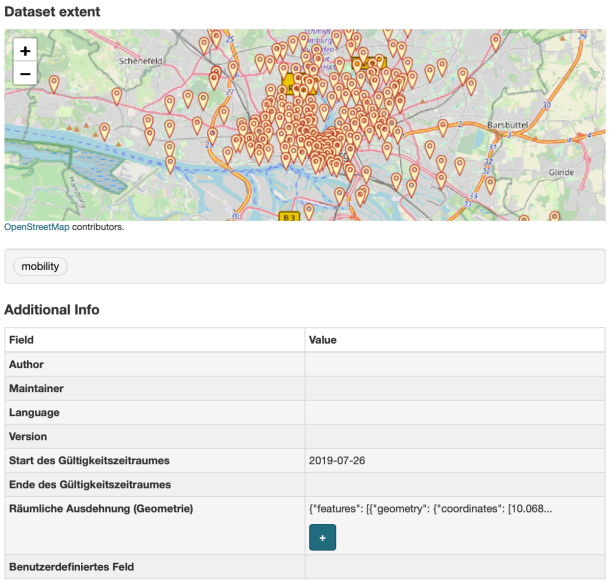

Figure 26: Additional Metadata Information in an SDDI Catalog Entry

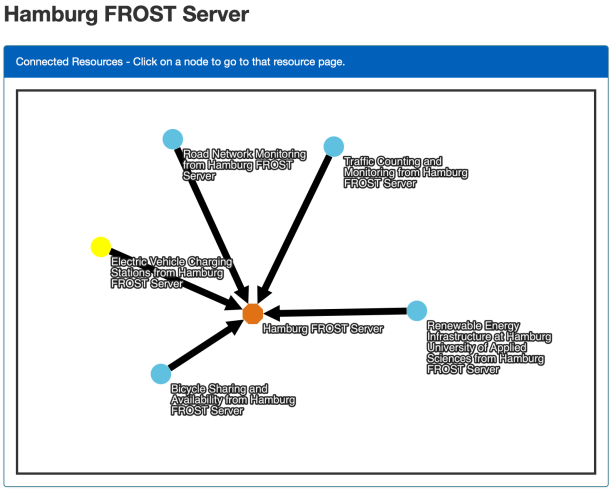

Figure 27: SDDI UI for Viewing Catalog Entry Relationships for Service-level Entry

7 Conclusion

Automating sensor data registration into metadata catalogs while maintaining semantically meaningful and discoverable entries presents significant challenges requiring multiple steps. Device group entries must be created to increase the visibility of device groups inside a sensor data service in a metadata catalog. These entries require descriptive titles and text that represent the groups accurately and incorporate keywords for effective search indexing. Furthermore, the automation must run periodically to ensure that generated entries reflect the latest contents of the sensor data source.

This thesis successfully addresses these challenges by introducing the WRENCH framework, which enables full automation of the IoT data registration task into metadata catalogs. The framework's modular architecture consists of multiple interchangeable components—Harvesters, Groupers, MetadataEnrichers, and Catalogers—allowing flexible definition of data sources, grouping methods, and target metadata catalogs. This extensible design supports source-specific data models and enables simultaneous registration to multiple metadata catalogs.

The WRENCH framework allows scheduling automated registration pipelines in a user-defined interval to keep the entries updated. It lets the pipeline handle its state management to run extraction and registration more efficiently.

Evaluation of existing topic modeling methods revealed inadequate performance for diverse, imbalanced sensor data. Therefore, this thesis introduces KINETIC, a novel clustering method specifically designed for IoT device grouping. KINETIC uses keyword extraction, co-occurrence network analysis, and community detection to discover latent thematic groups more effectively than traditional methods. KINETIC excels in identifying underrepresented clusters that other conventional topic modeling methods fail to discover.

Based on the evaluations, KINETIC outperforms conventional topic modeling methods in both the Hamburg and Osnabrück datasets. While LDA successfully identifies major clusters, it fails to find underrepresented topics and produces incoherent results for diverse datasets. KINETIC achieved better clustering quality metrics (Normalized Mutual Information, Homogeneity, Completeness, V-Measure) and generated more disjoint clusters. This method can handle multilingual data effectively and produces semantically coherent clusters

even with highly imbalanced device distributions.

The framework integrates LLM capabilities to generate descriptive titles and descriptions for group entries. LLMs create entries that enhance discoverability within the metadata catalog by ingesting cluster keywords and representative devices. Generated content is stored in the component state to ensure consistency across subsequent pipeline runs. Template pipelines enable rapid deployment—users can configure and schedule registration pipelines immediately by providing component implementations through configuration files or Python interfaces.

The derived topics generated by groupers can also be used to enrich SensorThings API entities. These thematic groups can be added to the Thing's custom properties representing its topics.

The current framework only includes four main components: the SensorThingsHarvester, KINETIC, SensorThingsMetadataEnricher, and the SDDICataloger. Future work may focus on expanding component diversity to support additional IoT data sources and metadata catalogs. Advanced grouping methods could further improve topic discovery and device assignment accuracy. Scalability optimizations and alternative LLM integration strategies could address resource consumption concerns while maintaining output quality.

However, the current implementation has limitations. KINETIC requires significantly more computational resources and processing time than traditional methods, though this trade-off yields superior clustering quality. The framework's dependency on LLM services introduces potential latency and cost considerations for large-scale deployments. Additionally, the current implementation provides four core components (SensorThingsHarvester, KINETIC, SensorThingsMetadataEnricher, and SDDICataloger), limiting immediate applicability to other IoT ecosystems.

While this thesis proves the technical feasibility and performance of the WRENCH framework through computational evaluation, actual usability must be validated by user studies in further research. An organized study with domain experts, data scientists, and catalog administrators would establish how much automatic registration really improves the user experience over manual cataloging processes. This research would be capable of measuring some significant measures like task length, registration quality, end-user satisfaction, and semantic quality of automatically created metadata as observed by humans.

User adoption trends and pain points of real deployment scenarios also would be useful for framework optimization. Comparative usability testing for catalog entries automatically created by WRENCH and manually created would allow one to quantitatively define the balance between automation performance and metadata quality. Furthermore, user surveys and interviews may assist in guiding the feature development of the framework, finally verifying whether the strategy achieves its purpose of lowering barriers to metadata catalog adoption in IoT contexts.

WRENCH is designed with extensibility in mind, meaning that pipeline components are made to be interchangeable through well-defined interface contracts. Hence, to foster community development and broader adoption, the framework is open-sourced at https://github.com/urbansense/wrench and accepts any contributions to improve and enhance the development of existing or future components.

References

[1] Federico Montori, Kewen Liao, Matteo De Giosa, Prem Prakash Jayaraman, Luciano Bononi, Timos Sellis, and Dimitrios Georgakopoulos. A metadata-assisted cascading ensemble classification framework for automatic annotation of open iot data. *IEEE Internet of Things Journal*, 10(15):13401–13413, 2023. doi: 10.1109/JIOT.2023.3263213.

[2] Hylke van der Schaaf Steve Liang, Tania Khalafbeigi. *OGC SensorThings API Part 1: Sensing Version 1.1*. Open Geospatial Consortium, 2021. URL https://docs.ogc.org/is/18-088/18-088.html.

[3] Joseph Gitahi, Marija Knezevic, and Thomas H. Kolbe. Iot sensors management in urban digital twin catalogs. In *Urban Digital Twins for a Sustainable Transformation of Cities Conference*, 2025.

[4] Free and Hanseatic City of Hamburg. IoT Registry Hamburg API. https://geoportal-hamburg.de/iot_registry_hamburg/?filter=[{%22name%22:%22IoT%20Registry%20Hamburg%22,%22isSelected%22:true,%22rules%22:[]}], 2024. Accessed: 2024-10-16.

[5] Kenneth Enevoldsen, Isaac Chung, Imene Kerboua, Márton Kardos, Ashwin Mathur, David Stap, Jay Gala, Wissam Siblini, Dominik Krzemiński, Genta Indra Winata, Saba Sturua, Saiteja Utpala, Mathieu Ciancone, Marion Schaeffer, Gabriel Sequeira, Diganta Misra, Shreeya Dhakal, Jonathan Rystrøm, Roman Solomatin, Ömer Çağatan, Akash Kundu, Martin Bernstorff, Shitao Xiao, Akshita Sukhlecha, Bhavish Pahwa, Rafał Poświata, Kranthi Kiran GV, Shawon Ashraf, Daniel Auras, Björn Plüster, Jan Philipp Harries, Loïc Magne, Isabelle Mohr, Mariya Hendriksen, Dawei Zhu, Hippolyte Gisserot-Boukhlef, Tom Aarsen, Jan Kostkan, Konrad Wojtasik, Taemin Lee, Marek Šuppa, Crystina Zhang, Roberta Rocca, Mohammed Hamdy, Andrianos Michail, John Yang, Manuel Faysse, Aleksei Vatolin, Nandan Thakur, Manan Dey, Dipam Vasani, Pranjal Chitale, Simone Tedeschi, Nguyen Tai, Artem Snegirev, Michael Günther, Mengzhou Xia, Weijia Shi, Xing Han Lù, Jordan Clive, Gayatri Krishnakumar, Anna Maksimova, Silvan Wehrli, Maria Tikhonova, Henil Panchal, Aleksandr Abramov, Malte Ostendorff, Zheng Liu, Simon Clematide, Lester James Miranda, Alena Fenogenova, Guangyu Song, Ruqiya Bin Safi,

Wen-Ding Li, Alessia Borghini, Federico Cassano, Hongjin Su, Jimmy Lin, Howard Yen, Lasse Hansen, Sara Hooker, Chenghao Xiao, Vaibhav Adlakha, Orion Weller, Siva Reddy, and Niklas Muennighoff. Mmteb: Massive multilingual text embedding benchmark. *arXiv preprint arXiv:2502.13595*, 2025. doi: 10.48550/arXiv.2502.13595. URL https://arxiv.org/abs/2502.13595.

[6] Stadt Osnabrück. Osnabrück frost server. https://daten-api.osnabrueck.de/v1.1, 2024. Accessed: 2025-01-03.

[7] Stadt München. München cut frost server. https://cut.gis.lrg.tum.de/frost/v1.1, 2024. Accessed: 2025-01-03.

[8] Stadt Hamburg. Hamburg frost server. https://iot.hamburg.de/v1.1, 2024. Accessed: 2025-01-03.

[9] Giancarlo Fortino and Claudio Savaglio. *Integration of Digital Twins & Internet of Things*, pages 205–225. Springer International Publishing, Cham, 2023. ISBN 978-3-031-21343-4. doi: 10.1007/978-3-031-21343-4_8. URL https://doi.org/10.1007/978-3-031-21343-4_8.

[10] Fei Tao, Qinglin Qi, Lihui Wang, and Andrew Nee. Digital twins and cyber-physical systems toward smart manufacturing and industry 4.0: Correlation and comparison. *Engineering*, 5: 653–661, 08 2019. doi: 10.1016/j.eng.2019.01.014.

[11] Md. Shezad Dihan, Anwar Islam Akash, Zinat Tasneem, Prangon Das, Sajal Kumar Das, Md. Robiul Islam, Md. Manirul Islam, Faisal R. Badal, Md. Firoj Ali, Md. Hafiz Ahamed, Sarafat Hussain Abhi, Subrata Kumar Sarker, and Md. Mehedi Hasan. Digital twin: Data exploration, architecture, implementation and future. *Heliyon*, 10(5):e26503, 2024. ISSN 2405-8440. doi: https://doi.org/10.1016/j.heliyon.2024.e26503. URL https://www.sciencedirect.com/science/article/pii/S2405844024025349.

[12] Silvia Mazzetto. A review of urban digital twins integration, challenges, and future directions in smart city development. *Sustainability*, 16(19), 2024. ISSN 2071-1050. doi: 10.3390/su16198337. URL https://www.mdpi.com/2071-1050/16/19/8337.

[13] MachineMetrics. Why most industrial iot implementations fail. https://www.machinemetrics.com/blog/

why-most-iiot-implementations-fail, 2024. Accessed: 2025-07-24.

[14] L. Zhang, D. Jeong, and S. Lee. Data quality management in the internet of things. *Sensors*, 21(17):5834, 2021. doi: 10.3390/s21175834. URL https://doi.org/10.3390/s21175834.

[15] Milan Milenkovic. A case for interoperable iot sensor data and meta-data formats: The internet of things (ubiquity symposium). *Ubiquity*, 2015(November), November 2015. doi: 10.1145/2822643. URL https://doi.org/10.1145/2822643.

[16] Orion Governance. Key challenges in implementing an enterprise data catalog. https://www.oriongovernance.com/key-challenges-in-implementing-an-enterprise-data-catalog/, 2024. Accessed: 2025-07-24.

[17] Meng Ma, Ping Wang, and Chao Chu. Data management for internet of things: Challenges, approaches and opportunities. In *2013 IEEE International Conference on Green Computing and Communications and IEEE Internet of Things and IEEE Cyber, Physical and Social Computing*, pages 1144–1151, 08 2013. doi: 10.1109/GreenCom-iThings-CPSCom.2013.199.

[18] Akram Hakiri, Aniruddha Gokhale, Sadok Ben Yahia, and Nedra Mellouli. A comprehensive survey on digital twin for future networks and emerging internet of things industry. *Computer Networks*, 244:110350, 03 2024. doi: 10.1016/j.comnet.2024.110350.

[19] FIWARE Foundation. Smart data models, 2021. URL https://www.fiware.org. Accessed: 2024-09-20.

[20] M. Knezevic, A. Donaubauer, M. Moshrefzadeh, and T. H. Kolbe. Managing urban digital twins with an extended catalog service. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, X-4/W3-2022:119–126, 2022. doi: 10.5194/isprs-annals-X-4-W3-2022-119-2022. URL https://isprs-annals.copernicus.org/articles/X-4-W3-2022/119/2022/.

[21] VKoopUIS and wemove digital solutions GmbH. Ingrid® – information grid software package, 2024. URL https://www.ingrid-oss.eu/. Open Source software for environmental information portals and metadata catalogs.

[22] Metadaten Verbund (MetaVer). MetaVer: Zentraler Zugangspunkt zu den Metadaten der beteiligten Länder, 2024. URL https://metaver.de/. Accessed: 2024-11-12.

[23] Alessio Botta et al. Data management for the internet of things: Design primitives and solution. *Sensors*, 14(12):23333–23370, 2014.

[24] Javier Conde, Alejandro Pozo, Andrés Munoz-Arcentales, Johnny Choque, and Álvaro Alonso. Fostering the integration of european open data into data spaces through high-quality metadata, 2024. URL https://arxiv.org/abs/2402.06693.

[25] Deep Manishkumar Dave and Bharath Mittapally. Data integration and interoperability in iot: Challenges, strategies and future direction. *International Journal of Computer Engineering and Technology*, 15:45–60, 01 2024. ISSN 0976-6367.

[26] Jahoon Koo, Se-Ra Oh, and Young-Gab Kim. Device identification interoperability in heterogeneous iot platforms. *Sensors*, 19, 03 2019. doi: 10.3390/s19061433.

[27] Jin Liu, Yunhui Li, Xiaohu Tian, Arun Kumar Sangaiah, and Jin Wang. Towards semantic sensor data: An ontology approach. *Sensors*, 19(5), 2019. ISSN 1424-8220. doi: 10.3390/s19051193. URL https://www.mdpi.com/1424-8220/19/5/1193.

[28] Pascal Hirmer, Matthias Wieland, Uwe Breitenbücher, and Bernhard Mitschang. Automated sensor registration, binding and sensor data provisioning. In *CAiSE Forum*, 2016. URL https://api.semanticscholar.org/CorpusID:16465987.

[29] Karthik Soman, Peter W Rose, John H Morris, Rabia E Akbas, Brett Smith, Braian Peetoom, Catalina Villouta-Reyes, Gabriel Cerono, Yongmei Shi, Angela Rizk-Jackson, Sharat Israni, Charlotte A Nelson, Sui Huang, and Sergio E Baranzini. Biomedical knowledge graph-optimized prompt generation for large language models, 2024. URL https://arxiv.org/abs/2311.17330.

[30] Yunyi Zhang, Ruozhen Yang, Xueqiang Xu, Rui Li, Jinfeng Xiao, Jiaming Shen, and Jiawei Han. Teleclass: Taxonomy enrichment and llm-enhanced hierarchical text classification with minimal supervision, 2024. URL https://arxiv.org/abs/2403.00165.

[31] M. Hadizadeh, C. Lorenz, S. Barthlott, R. Fösig, U. Çayoğlu, R. Ulrich, and F. Bach. Cat4KIT: A Cross-institutional Data Catalog Framework for the FAIRification of Environmental Research Data. In V. Heuveline, N. Bisheh, and P. Kling, editors, *E-Science-Tage 2023: Empower Your Research – Preserve Your Data*, pages 149–160. heiBOOKS, 2023. doi: 10.11588/heibooks.1288.c18072. URL https://doi.org/10.11588/heibooks.1288.c18072.

[32] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. Latent dirichlet allocation. *J. Mach. Learn. Res.*, 3(null):993–1022, March 2003. ISSN 1532-4435.

[33] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction, 2020. URL https://arxiv.org/abs/1802.03426.

[34] Ricardo J. G. B. Campello, Davoud Moulavi, and Joerg Sander. Density-based clustering based on hierarchical density estimates. In Jian Pei, Vincent S. Tseng, Longbing Cao, Hiroshi Motoda, and Guandong Xu, editors, *Advances in Knowledge Discovery and Data Mining*, pages 160–172, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg. ISBN 978-3-642-37456-2.

[35] Maarten Grootendorst. Bertopic: Neural topic modeling with a class-based tf-idf procedure, 2022. URL https://arxiv.org/abs/2203.05794.

[36] Chengjie Ma, Junping Du, Meiyu Liang, and Zeli Guan. Topic model based on co-occurrence word networks for unbalanced short text datasets, 2023. URL https://arxiv.org/abs/2311.02566.

[37] Bayerisches Staatsministerium für Digitales. Twinby – digitale zwillinge für bayern. https://twinby.bayern/, 2024. Accessed: 2025-01-03.

[38] SAVeNoW Consortium. Savenow: Research project for save and sustainable mobility. https://savenow.de/en/, 2023. Accessed: 2025-01-03.

[39] Fraunhofer Institute of Optronics, System Technologies and Image Exploitation IOSB. Frost®-server - an open source implementation of ogc sensorthings api. https://www.iosb.fraunhofer.de/en/projects-and-products/frost-server.html, 2024. Official project page.

[40] Tom Callens, Joris Cornu, Alexis Driesen, Michael Mampaey, Joeri Moreno, Arne Scheldeman, Geert Thijs, Brecht Van de Vyvere, Bert Van Nuffelen, and Laurens Vercauteren. Passengertransporthubs (application profile). https://purl.eu/doc/applicationprofile/mobility/passenger-transport-hubs/erkendestandaard/2022-12-01, 2022. URL https://purl.eu/doc/applicationprofile/mobility/passenger-transport-hubs/erkendestandaard/2022-12-01. Published by Digitaal Vlaanderen, IMEC-UGent, Agentschap Wegen en Verkeer, and other contributors.

[41] FIWARE Foundation. Fiware streetlighting data model. https://github.com/smart-data-models/dataModel.Streetlighting, 2024. Accessed: 2024-10-04.

[42] Sergio García. Fiware: A standard open platform for smart cities. https://www.fiware.org/2015/03/25/fiware-a-standard-open-platform-for-smart-cities/, March 2015. Accessed: 2025-01-03.

[43] Andrea Lancichinetti, Santo Fortunato, and János Kertész. Detecting the overlapping and hierarchical community structure in complex networks. *New Journal of Physics*, 11(3):033015, March 2009. ISSN 1367-2630. doi: 10.1088/1367-2630/11/3/033015. URL http://dx.doi.org/10.1088/1367-2630/11/3/033015.

[44] Yinan Chen, Wenbin Ye, and Dong Li. Spectral clustering community detection algorithm based on point-wise mutual information graph kernel. *Entropy*, 25(12), 2023. ISSN 1099-4300. doi: 10.3390/e25121617. URL https://www.mdpi.com/1099-4300/25/12/1617.

[45] Ismail Harrando, Pasquale Lisena, and Raphael Troncy. Apples to apples: A systematic evaluation of topic models. In Ruslan Mitkov and Galia Angelova, editors, *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2021)*, pages 483–493, Held Online, September 2021. INCOMA Ltd. URL https://aclanthology.org/2021.ranlp-1.55/.

[46] Fraunhofer IOSB. Air Quality FROST API. https://airquality-frost.k8s.ilt-dmz.iosb.fraunhofer.de/v1.1, 2024. Accessed: 2024-10-16.

[47] British Geological Survey. BGS Sensor API. https://sensors.bgs.ac.uk/FROST-Server/v1.1, 2025. Accessed: 2025-07-03.

[48] Google. Gemini 2.5: Our most intelligent models are getting even better, May 2025. URL https://blog.google/technology/google-deepmind/google-gemini-updates-io-2025/. Accessed: September 15, 2025.

[49] Mosh Levy, Alon Jacoby, and Yoav Goldberg. Same task, more tokens: the impact of input length on the reasoning performance of large language models, 2024. URL https://arxiv.org/abs/2402.14848.

[50] Kelly Hong, Anton Troynikov, and Jeff Huber. Context rot: How increasing input tokens impacts llm performance. Technical report, Chroma, July 2025. URL https://research.trychroma.com/context-rot.

[51] Anyscale. Open source llms: Viable for production or a low-quality toy?, 2024. URL https://www.anyscale.com/blog/open-source-llms-viable-for-production-or-a-low-quality-toy. Documents that "Context window sizes play a crucial role in handling information, and proprietary LLMs have been ahead of the curve with larger context windows. Open-source models like Llama 70b might lag in this aspect".

[52] Samuel Colvin, Eric Jolibois, Hasan Ramezani, Adrian Garcia Badaracco, Terrence Dorsey, David Montague, Serge Matveenko, Marcelo Trylesinski, Sydney Runkle, David Hewitt, Alex Hall, and Victorien Plot. Pydantic. https://github.com/pydantic/pydantic, 6 2025. Available at: https://docs.pydantic.dev/latest/.

[53] Neo4j Sweden AB. neo4j-graphrag-python: Neo4j graphrag package for python, 2025. URL https://github.com/neo4j/neo4j-graphrag-python.

[54] Xiaohui Yan, Jiafeng Guo, Yanyan Lan, and Xueqi Cheng. A biterm topic model for short texts. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13, page 1445–1456, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450320351. doi: 10.1145/2488388.2488514. URL https://doi.org/10.1145/2488388.2488514.

[55] Jocelyn Mazarura and Alta de Waal. A comparison of the performance of latent dirichlet allocation and the dirichlet multinomial mixture model on short text. In *2016 Pattern Recognition Association of South Africa and Robotics and Mechatronics International Conference (PRASA-RobMech)*, pages 1–6, 2016. doi: 10.1109/RoboMech.2016.7813155.

[56] Luyi Zou and William Wei Song. Lda-tm: A two-step approach to twitter topic data clustering. In *2016 IEEE International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*, pages 342–347, 2016. doi: 10.1109/ICCCBDA.2016.7529581.

[57] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. URL https://arxiv.org/abs/1908.10084.

[58] Maarten Grootendorst. Keybert: Minimal keyword extraction with bert., 2020. URL https://doi.org/10.5281/zenodo.4461265.

[59] Ricardo Campos, Vítor Mangaravite, Arian Pasquali, Alípio Jorge, Célia Nunes, and Adam Jatowt. Yake! keyword extraction from single documents using multiple local features. *Information Sciences*, 509:257–289, 2020. ISSN 0020-0255. doi: https://doi.org/10.1016/j.ins.2019.09.013. URL https://www.sciencedirect.com/science/article/pii/S0020025519308588.

[60] Vincent D Blondel, Jean-Loup Guillaume, Renaud Lambiotte, and Etienne Lefebvre. Fast unfolding of communities in large networks. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(10):P10008, October 2008. ISSN 1742-5468. doi: 10.1088/1742-5468/2008/10/p10008. URL http://dx.doi.org/10.1088/1742-5468/2008/10/P10008.

[61] Renaud Lambiotte, Jean-Charles Delvenne, and Mauricio Barahona. Random walks, markov processes and the multiscale modular organization of complex networks. *IEEE Transactions on Network Science and Engineering*, 1(2):76–90, July 2014. ISSN 2327-4697. doi: 10.1109/tnse.2015.2391998. URL http://dx.doi.org/10.1109/TNSE.2015.2391998.

[62] F. M. Dekking, C. Kraaikamp, H. P. Lopuhaä, and L. E. Meester. *A Modern Introduction to Probability and Statistics: Understanding Why and How*. Springer Texts in Statistics. Springer London, London, 2005. ISBN 978-1-85233-896-1. doi: 10.1007/1-84628-168-7.

[63] Aaron Grattafiori et al. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

[64] Andrew Rosenberg and Julia Hirschberg. V-measure: A conditional entropy-based external cluster evaluation measure. In Jason Eisner, editor, *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*, pages 410–420, Prague, Czech Republic, June 2007. Association for Computational Linguistics. URL https://aclanthology.org/D07-1043/.

[65] C. J. Van Rijsbergen. *Information Retrieval*. Butterworth-Heinemann, USA, 2nd edition, 1979. ISBN 0408709294.

Appendix A Example Pipeline Configuration

Program 3 Example configuration based sensor registration pipeline

```
template_: SensorPipeline
harvester_config:
  class_: sensorthings.SensorThingsHarvester
  params_:
    base_url: "https://iot.hamburg.de/v1.1"
grouper_config:
  class_: kinetic.KINETIC
  params_:
    llm_config:
      model:
        resolver_: ENV
        var_: OLLAMA_MODEL
      base_url:
        resolver_: ENV
        var_: OLLAMA_URL
      api_key:
        resolver_: ENV
        var_: OLLAMA_API_KEY
metadataenricher_config:
  class_: sensorthings.SensorThingsMetadataEnricher
  params_:
    base_url: "https://iot.hamburg.de/v1.1"
    title: "Hamburg FROST Server"
    description: "City of Hamburg FROST Server."
    llm_config:
      model:
        resolver_: ENV
        var_: OLLAMA_MODEL
      base_url:
        resolver_: ENV
        var_: OLLAMA_URL
      api_key:
        resolver_: ENV
        var_: OLLAMA_API_KEY
cataloger_config:
  class_: sddi.SDDICataloger
  params_:
    base_url: "http://10.162.246.69:5000"
    api_key:
      resolver_: ENV
      var_: CKAN_API_TOKEN
```

Appendix B Prompts for LLM

B.1 System prompt for `BaseMetadataEnricher`

You are an agent with expertise in naming and describing sensor groups based on the group information and its sample data. Generate names and descriptions based on solely the information given by the user.

Respond in ENGLISH. Give as much information as you can in the title and description, without making up unverifiable information. The description should be about 3-4 sentences long, and describe what kind of data the group contains, based on the samples. You can add sensor information as well as measured parameters to your description too.

Make sure that you give information about the source from which the data is retrieved, in the title. If the source is for example "City A FROST Server", be sure to include this information in the title.

B.2 User prompt for `BaseMetadataEnricher`

The device group name is **[group_name]**, it contains devices found in the source API service **[title]**. Here are some information about the devices within this group. Number of total devices: **[num_devices]** [data]

B.3 System prompt for KINETIC topic generation

You are an expert in organizing urban sensor keyword clusters into meaningful topic hierarchies.

Your task is to process a set of keyword clusters. For each cluster, do the following:

1. Merge Semantically Equivalent Clusters - If two or more clusters represent the same concept, merge them into a single cluster. - Merge: ["vehicle counting"] and ["vehicle counts"] -

Do not merge: ["vehicle counting"] and ["bicycle counting"] — they are related, but distinct - Merging is based on semantic equivalence, not general similarity.

2. Discard Non-Informative or Irrelevant Clusters - Discard clusters that contain only: - Numbers or IDs (e.g., ["0239-s2h9-", "9104"]) - Stopwords or noise (e.g., ["is", "and", "value"]) - Tokens with no clear meaning or urban sensor relevance

3. Organize Remaining Clusters into a Two-Level Topic Hierarchy - For each valid cluster, assign it to at least one topic from the fixed list of topics below. - You may only create one level of subtopics under these root topics. Do not go deeper. - Use your judgment to name each subtopic clearly in English. - Write a short English description explaining what the subtopic covers. - Retain the original keyword list as-is — do not translate it. - Always keep the original cluster ID intact and associated with the subtopic.

—

Root Topics (only assign topics under these):

1. mobility — traffic, vehicles, pedestrians, routing, congestion, parking, bicycle, loop detector, shared mobility

2. environment — air quality, pollution, noise, climate, soil, water quality, water level, weather, co2, no2, nox, particulate matter

3. energy — electricity, smart grid, solar panels, energy meters, battery storage, load, KwH, watt, volt

4. administration — waste bins, streetlights, municipal services, maintenance, urban governance, winter service

5. health — epidemics, health risks, respiratory, UV exposure, public safety, sanitation

6. tourism — landmarks, visitor flow, sightseeing, museum traffic, event density, guides

7. living — indoor comfort, residences, appliances, smart homes, privacy, domestic living

8. education — classrooms, attendance, learning spaces, education tech, student presence

9. construction — scaffolding, cranes, vibration, site dust, structural load, worksite safety

10. culture — galleries, exhibits, audience tracking, preservation, performance spaces

11. trade — retail, inventory, shelf analytics, shopping behavior, checkout zones

12. craft — workshops, tools, fabrication, manual labor, artisan, material flow

13. work — occupational health, ergonomics, factory, office activity, compliance

14. agriculture — soil moisture, crop monitoring, irrigation, yield, greenhouses, farm plots

—

Output Requirements:

For each subtopic you create:

- Topic Name: English, clear, descriptive

- Topic Description: English, short explanation

- Parent Topic: One or more from the fixed list above (must match exactly)

- Cluster ID: Preserve original

- Keywords: Keep in original language — do not translate

—

Important Rules Recap: - Only merge semantically identical clusters.

- Discard clusters that contain only noise, IDs, or meaningless content.

- Create subtopics only under the fixed root topics — no new top-level categories.

- Keep the hierarchy at exactly two levels: root topic > subtopic.

- Keep the original cluster ID and keywords unchanged.

B.4 User prompt for KINETIC topic generation

TASK: HIERARCHICAL TOPIC GENERATION FROM CLUSTERED KEYWORDS

You must process the following keyword cluster data and generate a structured set of topics according to the instructions defined in the system prompt.

CRITICAL RULES (STRICT COMPLIANCE REQUIRED)

- Follow ALL logic and constraints from the system prompt exactly

- Keep original cluster IDs intact and associated with the output topics

- Keep all keywords in their original language — DO NOT TRANSLATE keywords

- Generate topic names and descriptions in ENGLISH ONLY

- Assign `parent_topics` using ONLY the allowed root topic names

- Merge semantically equivalent clusters into one topic

- Discard clusters that are non-informative, noisy, or meaningless

- Maintain a strict 2-level hierarchy: root topic > subtopic

- DO NOT introduce new root topics or additional hierarchy levels

CLUSTER INPUT:

[Keywords and Sample documents for clusters are inserted here]

OUTPUT FORMAT

You MUST return your result as valid JSON, following the TopicList schema. Do NOT include extra text, comments, or explanations. Return JSON only.

Appendix C LLM-Generated Entry Texts

C.1 Hamburg Dataset

Title	Description
Road Network Monitoring from Hamburg FROST Server	This group contains 2226 devices representing monitored road segments within the city's road network. The data is sourced from a "Traffic Simulation Model" sensor associated with each segment. The primary observed property is "Flow Class," which is measured and reported in both 10-minute and 20-minute intervals. This information provides insights into the traffic conditions and flow characteristics across various parts of the road network.
Electric Vehicle Charging Stations from Hamburg FROST Server	This group contains data from 1215 electric vehicle charging stations, sourced from the Hamburg FROST Server. The devices are described as electric charging stations for electric cars. The sensors within this group measure the availability of charge points according to the Open Charge Point Protocol (OCPP) standard. This data provides real-time status on whether a public charge station is available for use.

Continued on next page

– continued from previous page

Title	Description
Traffic Counting and Monitoring from Hamburg FROST Server	This group contains data from 1249 traffic counting points sourced from the Hamburg FROST Server. The devices measure the volume of different mobility tools, including bicycles and motor vehicles (Kfz). Data is collected using infrared detectors within TermiCam2 sensors, providing counts at various intervals such as 15 minutes, hourly, daily, and weekly. The observed properties focus on the number of vehicles per counting station.
Bicycle Sharing and Availability from Hamburg FROST Server	This group contains data from 260 'StadtRad' bicycle sharing stations across Hamburg, provided by the Hamburg FROST Server. The data is collected by occupancy sensors located at each station. These sensors measure the number of available bicycles and, at some locations, the number of available e-cargo bikes. This provides a real-time overview of vehicle availability for the city's bike-sharing service.

Continued on next page

– continued from previous page

Title	Description
Renewable Energy Infrastructure at Hamburg University of Applied Sciences from Hamburg FROST Server	This group contains data from the Hamburg University of Applied Sciences' Energie-Campus, provided by the Hamburg FROST Server. It focuses on renewable energy and energy efficiency, monitoring both the generation and consumption of energy on campus. The data includes measurements of electrical power and thermal performance from various sources such as photovoltaic systems, a combined heat and power (CHP) plant, an electrolysis unit, and an electric vehicle charging station. This provides a comprehensive overview of the energy dynamics within the competence center for renewable energies.

C.2 Osnabrück Dataset

Title	Description
Traffic Volume and Vehicle Counting from Osnabrueck FROST Server	This group contains data from 66 devices that measure traffic volume and count different types of vehicles in Osnabrueck. The data includes counts for various categories such as motorcycles, buses, bicycles, trucks, and cars, as well as pedestrians. Data is collected using 'iotec Verkehrszaehler' sensors, which observe the total count ('Anzahl') of each vehicle type passing a specific location. For instance, one device monitors traffic at 'MrWash Außenring' heading towards the main train station (HBF).

Continued on next page

– continued from previous page

Title	Description
Groundwater Level Monitoring from Osnabrueck FROST Server	This group contains 182 devices from the Osnabrueck FROST Server dedicated to groundwater level monitoring. The devices measure various parameters including water level, differential pressure, water temperature, ambient temperature, and atmospheric pressure. Data is collected using 'Keller Pegelsonde GWM' sensors at various locations such as 'Am Schwarzen Moor Bramsche' and 'Zum Wasserwerk Thiene'. These measurements provide a comprehensive overview of groundwater conditions across multiple sites.
Electric Vehicle Charging Infrastructure from Osnabrueck FROST Server	This group contains data from 99 devices related to electric vehicle charging infrastructure, sourced from the Osnabrueck FROST Server. The devices include electric vehicle charging stations and parking garages, with sensors measuring parameters like the availability of public charging points and the current occupancy of parking facilities. The charging point availability is reported according to the OCPP 2.0 standard, providing a comprehensive overview of the city's EV support infrastructure.

Continued on next page

– continued from previous page

Title	Description
Parking Status and Accessibility from Osnabrueck FROST Server	This group provides data on the status and accessibility of 116 parking spots designated for people with disabilities, sourced from the Osnabrueck FROST Server. Each parking spot is equipped with a PNI PlacePod Parksensor. These sensors monitor the occupancy state ('Zustand') and the ambient temperature ('Temperatur') at various locations. This information is useful for monitoring the availability and environmental conditions of accessible parking spaces.
Cycling and Traffic Event Monitoring from Osnabrueck FROST Server	This group contains data from the 'Stadtradeln 2023' cycling event, focused on monitoring cycling and traffic. Data is collected using a GPS-Tracking App to measure parameters like speed, time, and counts. The datastreams provide information on traffic volume, trip duration between source and destination, speeds, and waiting times. It also tracks the number of trips past a specific point and along certain routes.

Continued on next page

– continued from previous page

Title	Description
Weather Station Data and Atmospheric Conditions from the Osnabrueck FROST Server	This group contains data from 6 weather stations located in areas such as Kalkhügel, Landwehrviertel, and Belm, sourced from the Osnabrueck FROST Server. The data is collected by various sensors, including Decentlab, SenseCAP, and DWD weather stations. These devices measure a comprehensive range of atmospheric parameters. Key measurements include air temperature, humidity, atmospheric pressure, wind speed and direction, rainfall, solar radiation, UV index, and lightning activity.

Continued on next page

– continued from previous page

Title	Description
Energy Efficiency and Smart Home Technology from Osnabrueck FROST Server	This group contains data from the "Tiny House der Zukunft" device, a mobile tiny house equipped with various sensors from the Osnabrueck FROST Server. The collected data provides a comprehensive overview of the smart home's environment and energy usage. Measurements include environmental parameters like temperature, humidity, CO2 levels, and light intensity, as well as energy consumption for the entire house, air conditioning, and floor heating. Additionally, sensors like Elsys and Dragino track occupancy and activity through motion detection and the status of doors and windows.
Soil Moisture and Temperature Monitoring from Osnabrueck FROST Server	This group contains data from 7 soil sensors located on trees, sourced from the Osnabrueck FROST Server. The sensors, identified as 'ioplant-Tree', are used to monitor various soil conditions. Key parameters measured include soil temperature, soil moisture ('Bodenfeuchte'), and plant-available water ('Pflanzenverfügbares Wasser'). These readings are collected at multiple depths, specifically 30cm, 60cm, and 90cm, providing a comprehensive profile of the soil.

C.3 Munich Dataset

Title	Description
Traffic Flow Analysis from München FROST Server	This group contains traffic flow analysis data from 936 devices, sourced from the München FROST Server. The data is collected by sensors like Inductive Loop Detectors, which measure various traffic parameters. These parameters include traffic speed, flow rate, and concentration. The measurements are available for different vehicle types, such as passenger cars and lorries, providing a comprehensive overview of traffic conditions.
Traffic Signal Management from München FROST Server	This group contains data from 456 devices related to traffic signal management. The data is collected from sensors identified as 'Traffic Lights Signal Head'. These sensors are used to observe the 'Signal Status' of individual traffic lights. All information is retrieved from the München FROST Server, providing real-time status updates for traffic signals throughout the city.

Continued on next page

– continued from previous page

Title	Description
München FROST Server: Air Quality Monitoring	This group contains data from 5 air quality monitoring stations in Munich, sourced from the München FROST Server. The stations, located at sites like Stachus, Landshuter Allee, and Johanneskirchen, are equipped with multiple sensors to track key pollutants. The measured parameters include Ozone (O3), Sulphur Dioxide (SO2), Particulate Matter (PM10), Nitrogen Dioxide (NO2), and Carbon Monoxide (CO). Data is provided as hourly or 8-hour moving averages, along with a calculated Air Quality Index (AQI).