UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Kaspar Kadalipp

Knowledge Graphs for Cataloging and Making Sense of Smart City Data

Master's Thesis (30 ECTS)

Supervisor: Pelle Jakovits, PhD

Tartu 2024

Knowledge Graphs for Cataloging and Making Sense of Smart City Data

Abstract:

Modern buildings and cities are equipped with a large number of devices with sensors that generate data. However, this data is often stored in a technical format that is more convenient for the sensors, making it difficult for humans to understand. This thesis deals with the challenge of interpreting the complex data generated by the numerous sensors, using the Tartu Cumulocity IoT platform dataset as a case study. To get an overview of the available data and identify issues with analyzing it further, the dataset was visualized as a simplified knowledge graph. In addition, a hierarchical topic model was created to capture the nuances of various smart city domains from the dataset.

Keywords:

Cumulocity IoT platform, smart city, topic model, knowledge graph, prompt engineering

CERCS:

P170 Computer science, numerical analysis, systems, control

Teemamudel targa linna andmetest ülevaate saamiseks

Lühikokkuvõte:

Kaasaegsetes hoonetes ja linnades on rohkelt seadmeid, mille andurid saadavad pidevalt välja andmeid. Need andmed jäädvustatakse sageli anduri spetiifilises tehnilises vormingus, mida on inimesel keeruline mõista. Käesolev lõputöö proovib leida viisi süsteemselt organiseerida andurite poolt välja saadetud keerukaid andmeid, kasutades juhtumiuuringuna Tartu linna Cumulocity IoT platvormi andmestikku. Olemasolevatest andmetest ülevaate saamiseks ja nende edasise analüüsimisega seotud probleemide tuvastamiseks visualiseeriti andmestik lihtsustatud teadmusgraafi kujul ning koostati hierarhiline teemamudel, mis suudab andmekogumist tuvastada anduritele vastavaid nutistu valdkondi.

Võtmesõnad:

Cumulocity IoT platvorm, nutistu, teemamudel, teadmusgraaf, tekstiloomemootori viiba koostamine

CERCS:

P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine

Contents

1 Introduction

Smart cities use data to enhance their management and to improve the standard of living for residents. A key part of a smart city is the use of sensors and Internet of Things (IoT) devices. The data collected from these devices provides valuable insights to city planners and managers, helping them make informed decisions. However, organizing this data can be a significant challenge due to the varying standards employed by different companies and datasets. This inconsistency makes it difficult to merge and analyze the data effectively.

The Cumulocity IoT platform offers tools to assist in collecting, processing, and analyzing sensor data. However, over the past eight years, while integrating smart city solutions with various companies and datasets for the city of Tartu, the limitations of the platform have become more apparent. As Cumulocity does not offer a comprehensive overview of the data being stored, a need has arisen to develop a system that can provide a precise dataset overview.

This thesis aims to address the challenges associated with developing a machine-learning model that can categorize smart city IoT devices based on the data they transmit. With the increasing amount of data being collected, it has become difficult to track what kind of data is being collected and how it's being used. By using a large language model to extract insights, the model aims to bring clarity to the different types and uses of the existing data.

In order to achieve the desired goal, a program must first be developed to retrieve the statistics and schema for the dataset using the Cumulocity API, since direct access to the cloud platform's database is not available. Cumulocity does not provide a good way to provide an overview of the amount and types of data being stored. To address this, a visualization tool in the form of a simplified knowledge graph was created. This graph provides an overview of the stored data and can be used to identify problems with the current data categorization and naming standards.

The thesis is structured as follows: Section 2 outlines the various types of data that are managed by the Cumulocity platform and outlines the platform's key limitations. Section 3 examines the challenges of managing diverse data formats and uses the Tartu Cumulocity dataset to illustrate problems with existing naming standards. Section 4 describes the expected model capabilities and explains the topic modeling technique BERTTopic. Section 5 outlines the process of creating a hierarchical topic model that can describe the kind of data being sent by smart city devices. Section 6 describes the process of requesting data through Cumulocity API, how data is visualized, and highlights the importance of establishing strong data standards.

2 Cumulocity IoT Platform

Cumulocity [1] is an Internet of Things (IoT) platform offered as a Platform as a Service (PaaS). This cloud-based service model enables businesses to leverage a comprehensive suite of IoT capabilities without requiring extensive infrastructure investment or maintenance. By providing a robust framework for connecting and managing devices, collecting data, and integrating various IoT services, Cumulocity tries to simplify the complexities associated with the vast IoT ecosystem. This chapter provides an overview of the Cumulocity IoT platform's main ways of storing and retrieving data.

2.1 Data Streams

Cumulocity inventory serves as a repository for all device-related data, including configurations, supported operations, connections, associated assets like vehicles, machines, buildings, and their structural hierarchies [2]. The data generated by IoT devices mainly takes two forms: events and measurements. These formats are the primary methods through which the platform captures and organizes information from the connected IoT devices, enabling users to monitor, analyze, and respond to various conditions and activities.

2.1.1 Event Repesentation

Events represent occurrences or changes in the state of an IoT device or system. These events are triggered when specific conditions or changes in device states, such as sensor readings or connectivity status, are detected. Once a condition is met, the platform creates an event detailing what happened, when, and which device was involved. These events can prompt notifications to users, such as alerts via email or SMS, or initiate automated actions, like shutting down a device or starting maintenance.

Table 1. API Request Body Parameters for Creating an Event

Parameter	Type	Description
source.id	string	Associated device identifier.
text	string	Description of an event.
time	date-time	Creation date and time.
type	string	Event group name.
* (custom fragment)	any	Any additional key-value pairs.

As detailed in Table 1, events generally include custom fragments to represent information. This means that the structure and content of events can vary significantly based on the specific requirements and context of the IoT application. By standardizing the core aspects of event data (source, type, time, etc.) while allowing for customization, IoT systems can adapt to new requirements or devices without requiring a complete overhaul of the event handling architecture.


```
{
  "source": {
        "id": "31074736"
  },
  "type": "com_ridango_validation",
  "text": "Validation",
  "com_ridango_validation_data": {
        "trip_id": "713058",
        "stop_sequence": "13",
        "line": "1",
        "product_id": "6088",
        "stop_code": "",
        "passenger_count": "1",
        "location": "822",
        "timestamp": "2024-03-30 23:26:37"
  },
  "com_ridango_trip_data": {
        "trip_short_name": "Nõlvaku - FI",
```

```
        "trip_id": "1583862",
        "direction_id": "B>A",
        "route_id": "130725",
        "arrival_time": "23:34:00",
        "turnr": "713058",
        "departure_time": "23:12:00"
    },
    "com_ridango_product_data": {
        "period": "2592000",
        "product_type": "ticket",
        "booklet_duration": "",
        "price": "7.67",
        "product_id": "6088",
        "name": "Tartu 30-day discount ticket for students and seniors",
        "units": "1"
    }
}
```


Figure 1. Ticket Validation Event

Consider the event described in Figure 1, which illustrates a passenger validation event in a public transportation system. It includes details such as the bus route, validated ticket type, and location of ticket validation. This event is comprised of three custom fragments.

2.1.2 Measurement Repesentation

Measurements represent quantifiable data collected from IoT devices. This can include various metrics such as temperature, humidity, pressure, or any other numerical data that sensors can measure and report. Measurements are typically time-series data recorded with a timestamp to track changes over time.

Table 2. API Body Parameters for Creating a Measurement

Parameter	Type	Description
`source.id`	string	Associated device identifier.
`time`	date-time	Creation date and time.
`type`	string	Measurement group name.
`<fragment>.<series>.unit`	string	Unit of a measurement.
`<fragment>.<series>.value`	number	Value of a measurement.
`* (custom fragment)`	any	Any additional key-value pairs.

The representation of measurements, as shown in Table 2, follows a structured approach using the `fragment` and `series` notation. While it's possible to add custom fragments to the measurements, this is generally discouraged as measurements are intended to primarily consist of numerical data organized through fragments and series.


```
{
  "source": {
    "id": "137262339"
  },
  "type": "c4t_metric",
  "current": {
  "energy_cons": {
    "total": {
      "unit": "kWh",
      "value": 3764410.018
    },
    "L1": {
      "unit": "Wh",
      "value": 536888725
    },
    "L2": {
      "unit": "Wh",
      "value": 3223323516
    },
    "L3": {
      "unit": "Wh",
```

```
      "value": 4197777
    }
  },

  "current": {
    "L1": {
      "unit": "A",
      "value": 12.344
    },
    "L2": {
      "unit": "A",
      "value": 14.251
    },
    "L3": {
      "unit": "A",
      "value": 11.363
    }
  },
}
```


Figure 2. Energy Consumption Measurement

The `fragment` refers to a broader category or aspect of the device's operational parameters. Within each fragment, the `series` further delineates the specific reported metric, allowing for a granular breakdown of the data. The example provided in Figure 2 demonstrates this approach with the `energy_cons` and `current` fragments serving as the broader categories. Within these fragments, the data is organized into four series fields: `total`, `L1`, `L2`, `L3`. This data refers to a measurement from a three-phase power system.

2.2 API Limitations

While the Cumulocity API [3] offers a comprehensive set of functionalities for managing IoT devices and data, there are certain limitations that users may encounter during development and implementation.

2.2.1 Event Schema

One significant limitation is the inability to query what kinds of events a device sends. Since event structure can vary a lot and with query the structure of events can hinder developers who need to understand event composition, including available fields, data types, and relationships, from effectively processing and utilizing event data within their applications.

It is possible to retrieve events for a specific device that has a custom fragment in an event object. However, if the custom fragment value is an object, then any nested keys cannot be used for queries. Custom fragments can only be queried based on the value, but only if the value is a string. This discourages storing complex events as custom fragments.


```
{
  "source": {
        "id": "321396631"
  },
  "type": "c8y_LocationUpdate",
  "text": "LocUpdate",
  "c8y_Position": {
        "lat": "26.7286635",
        "lng": "79.5999984741211",
        "alt": "58.3560662",
  }
}
```


Figure 3. Location Update Event

For the event in Figure 3, the API allows requesting all events for a device that uses `c8y_Position` custom fragment in an event, but it is not possible to request events that also have `lat`, `lon` or `alt` keys within the custom fragments.

2.2.2 Measurements Schema

Although the API allows querying schema for measurement fragments and series, there is no way to retrieve a list of utilized measurement types. While the data type may sometimes be disregarded, certain types, such as aggregated data, can significantly impact the results. The inability to readily identify such types within the API may lead to inaccuracies in data analysis and interpretation.

In a smart city, IoT devices are deployed to monitor air quality across various locations. These devices collect data on pollutants like particulate matter (PM2.5, PM10), nitrogen dioxide (NO2), sulfur dioxide (SO2), and carbon monoxide (CO). Suppose the city's environmental department aims to analyze this data to understand pollution trends and assess the effectiveness of pollution control measures. They rely on an API to query the data from these IoT devices. If the data is queried as a time series, then the measurement type isn't included in the result. Without knowing whether the retrieved data includes aggregated or normal measurements, analysts may inadvertently mix aggregated hourly averages. This oversight could potentially skew conclusions about pollution levels in different neighborhoods

2.2.3 Transmission Frequency

Another limitation is related to the visibility and management of device metadata, particularly regarding the frequency of data transmissions. This aspect is important for users who monitor device activity and ensure timely data collection. For instance, if a device is configured to send measurements at irregular intervals, Cumulocity does not inherently provide a straightforward method to discern the usual frequency of these transmissions. This limitation poses a challenge when establishing alerts to detect device inactivity or offline status, as some devices might send measurements every 10 seconds while others might send one measurement per day. So, users need to devise mechanisms or workarounds to track device activity without an easy way to access or interpret the

expected transmission patterns directly from the platform.

The Cumulocity API also lacks a direct method to query counts of events or measurements. To obtain this data, developers can employ a workaround by calling an endpoint for retrieving all events or measurements in the desired time frame, coupled with setting the pagination to a maximum of one item per page. Consequently, the total number of pages returned by the API corresponds to the count of measurements or events during the specified period.

2.2.4 Response Pagination

The API imposes specific limitations on data requests as well. Among these constraints, the maximum page size for data requests has a default value of 5 and is capped at 2000. This parameter defines the number of items that can be returned in a single request, and any request that returns more data is paginated. This limitation can be bypassed for measurements by requesting data in CSV format, but this option is not available for events. Among Tartu Cumulocity devices, a few have occasionally generated over a million events in a single month, while around a hundred devices have sometimes recorded more than 100,000 events monthly. Therefore, accounting for pagination is unavoidable when requesting data.

Table 3. Requesting Event Data with Default Pagination

Page Number	10,000	20,000	50,000	100,000	200,000
Response Time	5 seconds	10 seconds	25 seconds	50 seconds	100 seconds

Since pagination isn't dependent on content size, requesting many large events in one request can result in the request failing due to the payload being too large. However, sticking to the default page size also has its problems in scenarios where the requested data encompasses a large volume of events. As can be seen in Table 3, requests start taking much longer as the page number increases, and any requests that take longer than

two minutes get timed out. To mitigate these delays and optimize data retrieval, users would need to segment their requests by date. Although effective in reducing wait times, this approach introduces additional complexity and can be cumbersome.

3 Navigating Data Diversity

Smart cities process large volumes of data from various sources. To make use of this data, it needs to be properly managed. This chapter highlights the importance of proper data categorization and the complexities of managing diverse types, using the Tartu Cumulocity dataset as an example.

3.1 The Challenge of Diverse Data Formats

Recent studies have highlighted various challenges in implementing smart city technologies in the work of R. Jose and H. Rodrigues [4]. Similarly, future research directions in smart city initiatives have been explored in recent literature in the work of B. Ramdani and P. Kawalek [5]. Smart cities need to leverage data to analyze and improve the quality of life for their residents, aiming to meet the needs of present and future generations. By collecting and analyzing vast amounts of city data collected by various systems and sensors, authorities can gain valuable insights into ongoing trends, such as traffic patterns, air quality, energy consumption, and citizen engagement.

The success of smart city technologies depends on the accuracy and quality of the collected data. This involves integrating data from various sources and systems to gain a complete understanding of the city's operations. One of the main challenges is the lack of standardization, as different manufacturers and software developers use their own methods for data representation. An example of various data formats and fields can be found in the Estonian open data portal [6], where there are more than 1700 datasets from more than 2200 publishers. This issue becomes more complex when the available data doesn't follow standardized naming conventions. Examples of this include inconsistent names, ambiguous naming, allowing mistakes in names to propagate to newer data, and overuse of abbreviations. As a result, companies must invest significant resources in developing custom solutions for data normalization and integration.

3.2 Tartu Cumulocity Dataset

The dataset from the Tartu Cumulocity IoT platform illustrates the difficulties in managing diverse data formats. It collects data from different companies using various IoT devices and systems to monitor urban infrastructure such as traffic lights and public transportation systems. The absence of standardized data formats and naming conventions has led to a disorganized dataset, making it difficult to comprehend the types of data being collected.


```
{
  "device": "Tartu, Raekoja plats, 3 electricity.o",
  "type": "group",
  "fragment": "value",
  "series": "value",
  "unit": "kWh"
}, {
  "device": "sensor 1570 (SE 1584)",
  "type": "c4t_metric",
  "fragment": "sensor_125",
  "series": "particles_pm2_5_ug",
  "unit": "m"
}, {
  "device": "[keskkond] [Riia/Pepleri ristmik] suhteline õhuniiskus (%)",
  "type": "dal_series_target_measurement",
  "fragment": "dal",
  "series": "series",
  "unit": "%"
}, {
  "device": "ILM_Tartu_Õhuniiskus",
  "type": "dal_series_target_measurement",
  "fragment": "Dal",
  "series": "value",
  "unit": "percent"
}, {
  "device": "SEC_ET_Elekter akudest ja päikesest",
  "type": "dal_series_target_measurement",
  "fragment": "measure",
  "series": "item",
  "unit": "kWh"
}
```


Figure 4. Example of IoT Device Data from Tartu

The current device naming conventions, as shown in Figure 4, display several inconsistencies and poor practices that complicate data management. Ambiguous identifiers such as `sensor 1570 (SE 1584)` and non-descriptive tags like `sensor_125` do not provide enough contextual information. Additionally, to understand the data being sent, one often has to rely solely on the device name, as identifiers like `dal` and `series` do not offer any meaningful information. Furthermore, the use of special characters and brackets in identifiers like `[keskkond] [Riia/Pepleri ristmik]` may cause issues in data processing scripts due to potential misinterpretation by software requiring character escaping. Lastly, there is a problematic mix of Estonian and English in device names, which vary widely in style and detail. These issues highlight the urgent need for a systematic and standardized approach to naming conventions, ensuring the integrity of the database and improving its usability.

3.3 Cumulocity Sensor Library

The sensor library [7] is a collection of predefined fragments that, when included as part of the device's static metadata, help categorize and provide more control over incoming data. Utilizing one of the sensor library fragments enables the platform to automatically offer appropriate handling for a given type of data, such as data visualization and alerting based on predefined thresholds. For instance, a device defined as `c8y_TemperatureSensor` is expected to receive temperature readings with `c8y_TemperatureMeasurement` in the measurement structure. Because the data is identified as temperature data, it can be automatically converted between Celsius and Fahrenheit. However, there are relatively few such fragments, which necessitate custom configuration for devices.

3.4 Rule Based Categorzation

One way to tackle the issue is by creating a rule-based system for transforming data. This system can be developed with the assistance of a domain expert who can establish a set of business rules. These rules would enable automatic categorization or remapping of incoming data. However, a significant challenge lies in obtaining a comprehensive overview of the various types of data being transmitted to Cumulocity. This means that developing a custom solution would require more resources. Additionally, the lack of enforced standardization could negatively impact the effectiveness of any rule-based categorization system.

Cumulocity allows devices to be assigned to groups, but these groups cannot be used to query measurements or events of specific types. For instance, while all city building data devices can be grouped together, the temperature of all devices in this group cannot be directly queried. Instead, it is necessary to retrieve the IDs of all devices in the building group and then create separate temperature queries for each unique device ID.

3.5 Existing Solutions

Visualizing complex data can be achieved by creating a knowledge graph based on the dataset. A knowledge graph is a network graph that represents relationships between entities and is commonly used to depict data within a specific domain. Figure 5 depicts the structure of a knowledge graph, where nodes symbolize entities such as people, objects, or concepts, while edges illustrate the relationships between them. Graph management systems like Neo4j are often utilized to store graph data, allowing for both data storage and graph visualization to draw insights from the data. Interestingly, knowledge graphs can also be used to capture characteristics of complex concepts such as internet memes, as demonstrated in the work of R. Tommasini and T. Wijesiriwardene [8].

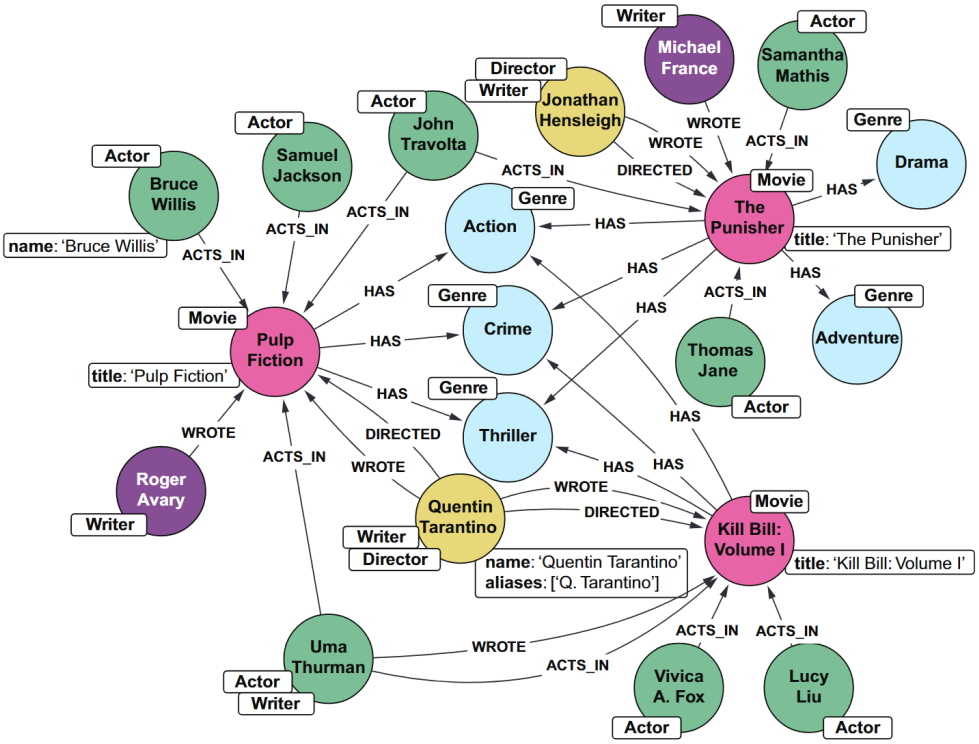

Figure 5. Knowledge Graph Representation for Three Movies [9]

Commercial applications like GraphAware Hume [10] can transform unknown documents into actionable knowledge graphs. They use large language models to process input data and create an initial meta-graph. Then, they enrich the output with domain knowledge and perform data validation. Next, machine learning techniques such as similarity and topic modeling are applied to construct the final knowledge graph. This knowledge graph can be used for different purposes, such as visualization, searching information, or as input to a machine learning model. GraphAware's approach is detailed in two of their published books, "Knowledge Graphs Applied" and "Graph-Powered Machine Learning" [9, 11]. They have also included advanced features in the knowledge graph, such as question prompts for navigation, which make it easier to filter the complex data.

4 Data Categorization Challenges

This chapter describes techniques and prerequisites needed for creating a machine learning model that could be used to extract insights from smart city IoT sensor data.

4.1 Model Selection Criteria

Selecting the appropriate machine learning model to address the challenges posed by a smart city IoT dataset involves considering several criteria.

4.1.1 Unsupervised and Semi-Supervised Learning Capabilities

Given the absence of a reliable labeled dataset for smart city IoT data, the primary criterion for model selection is the ability to perform well under unsupervised or semi-supervised conditions. Models designed for these learning paradigms can effectively discern structure and categorize data without extensive pre-labeled examples. Techniques such as clustering, association, and self-training can be valuable for extracting useful information from unlabeled data.

4.1.2 Natural Language Processing

The dataset contains both Estonian and English, so the model needs to have natural language processing (NLP) capabilities to handle multilingual data. Since the necessary information to correctly categorize a device is often embedded within device names, it's important to select a model that can understand the context and semantics of the data. Large language models are strong candidates as they are trained in multiple languages and have shown to be capable of understanding complex information.

4.2 Topic Modeling

Topic modeling is an unsupervised learning technique used to find abstract topics that occur in a collection of documents. It is widely used in natural language processing to uncover semantic patterns within text data, which then can be used for document clustering and information retrieval.

4.2.1 Fundamental Concepts

The main concept behind topic modeling is that documents consist of mixtures of topics, where a topic is a probability distribution over words. For text analysis, topic models can automatically organize and provide insights into large volumes of text data by grouping texts into thematically similar categories. This is valuable for summarizing and exploring large datasets, improving search engines, and tracking content trends.

One popular method for topic modeling is Latent Dirichlet Allocation (LDA), which assumes that each document is a mix of various topics, and each topic is a mix of words [12]. For example, a topic related to transportation could be identified by keywords such as "departures," "bus," and "traffic." Common words found in most input documents are disregarded as they do not provide specific information that distinguishes a document. However, this approach does not consider the semantics of the text.

4.2.2 Related Work

One approach to make sense of sensor data would be to create a topic model using information gathered from smart city-related publications, blog posts, and social network posts, similar to the approach by A. Kousis and C. Tjortjis in their work on smart city topic modeling [13]. This topic model could then be used to categorize devices. However, it is preferable to create a topic model based on actual data, as it would offer more detailed insights into the dataset rather than just the general smart city domain.

4.2.3 BERTTopic

To create a topic model that takes into account text semantics and context, topic modeling techniques like BERTTopic [14] can be used. In order to extract and refine topics from text BERTTopic employs a sequential five-step process, with an option sixth step [15]. The steps are as follows:

1. Text Embedding: The process starts by transforming input text into embeddings, converting it into a fixed-sized vector format while capturing semantics and context.

2. Dimensionality Reduction: To avoid the curse of dimensionality, text embeddings, which are often high-dimensional, undergo dimensionality reduction to simplify the data while preserving essential structures for better clustering.

3. Clustering: Using a clustering algorithm like HDBSCAN, the reduced embeddings are grouped into clusters based on their densities, with each cluster potentially representing a unique topic.

4. Topic Creation: Topics are formed from these clusters by assessing the density and distribution of documents within each cluster and using a bag-of-words representation to ensure that each topic is distinct and meaningful.

5. Topic Representation: For each identified topic, representative words are selected based on their c-TF-IDF scores, highlighting the most defining terms of each topic.

6. Refinement and Labeling: The final step involves refining the topics for coherence and, if necessary, labeling them. This can include merging similar topics, splitting broad ones, and eliminating outliers.

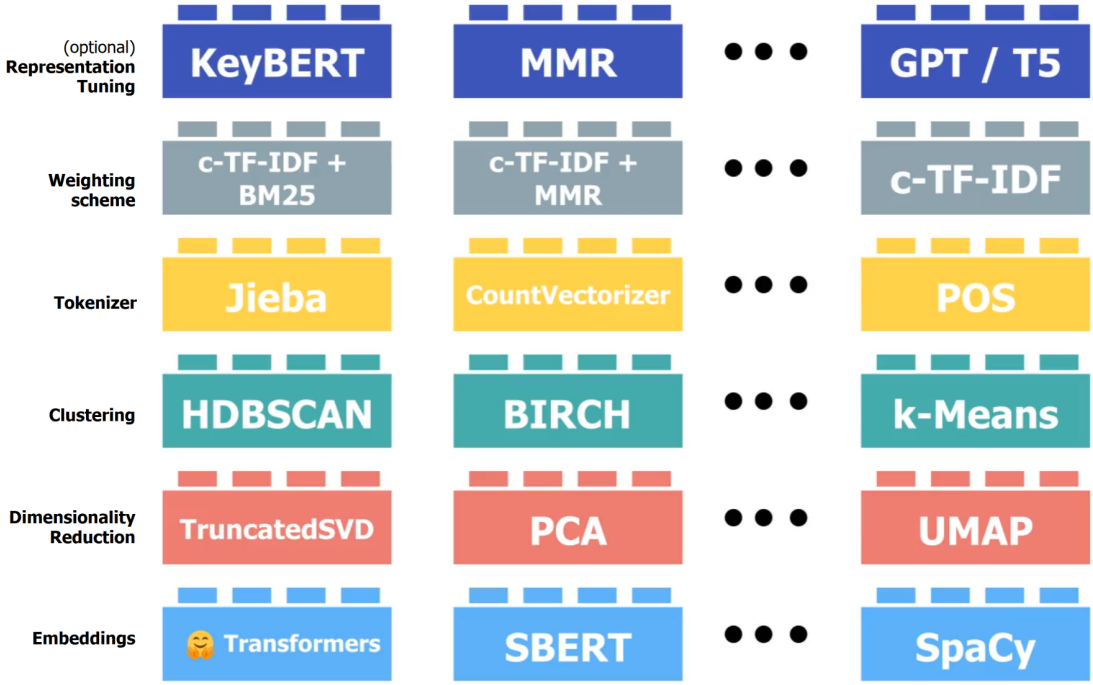

Figure 6. BERTTopic Modual Model [15]

The six steps in the process, illustrated in Figure 6, are designed to be independent of one another, allowing each component to be swapped out for any state-of-the-art machine-learning technique. This modular framework supports the development of customized topic models that can be modeled for specific domains and different languages.

5 Creating the Model

The objective was to create a topic model that can describe the different types of smart city IoT devices present in the Tartu Cumulocity dataset or similar datasets. This section outlines the process of augmenting the input data and creating a hierarchical topic model.

5.1 Problems with Input Data

To create the input data for the model, the device name and type were combined with a single instance of each unique measurement and event representation. When retrieving device information from the Cumulocity inventory, many devices include custom fragments in the configuration, such as longitude or latitude. However, this does not provide useful context for categorizing the type of data sent, so it was omitted from the input.

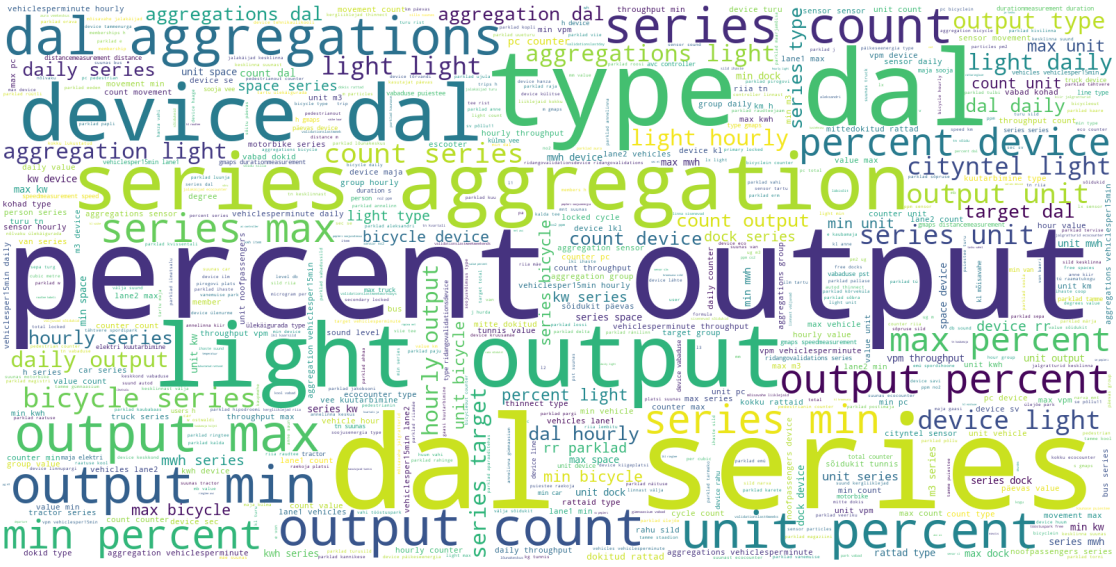

Figure 7. Word Cloud for Measurement Input Data

Creating a topic model limited to smart city domains from the initial dataset proved to be difficult due to the presence of information unrelated to the actual measurements. While text embedding can capture the semantic context of the text, it doesn't prevent

devices from being grouped based on recurring patterns in device names or common abbreviations like `dal`. Additionally, the dataset is unbalanced, with a large proportion of devices being street lights, as indicated by the `percent output` and `light output` keywords in the word cloud shown in Figure 7.

After experimenting with various sub-components for BERTTopic, it became clear that the most effective way to improve results was to extract useful features from the dataset before generating text embeddings for creating the topic model.

5.2 Utilizing Large Language Models

Large language models (LLMs) are trained on a vast amount of data, which includes information relevant to smart city domains, and they are able to understand natural language. An attempt was made to augment the input data by having LLMs classify devices into a predefined list of smart city domains. However, this approach also produced poor results.

5.2.1 Prompt Engineering

A different approach was necessary. Rather than instructing the LLM to categorize devices according to smart city domains, it was more effective to have the LLM describe the measurement data being transmitted. This approach would help filter the dataset to keep mostly relevant information related to measurements. Achieving the best results from LLMs requires the use of efficient engineering techniques.

```
 {
    "role": "system",
    "content": "Take on the persona of a data analyst who is proficient in
    ↪  interpreting JSON objects and extracting meaningful insights from them.
    ↪  The user will provide JSON objects representing measurement data from
    ↪  a smart city IoT device."
},
{
    "role": "user",
    "content": "Ignore device-specific information and concisely summarise
    ↪  what kind of data is being sent. 'Don't use references to time
    ↪  intervals such as hourly, daily, and monthly. Avoid generic terms like
    ↪  IoT and smart city. Also provide an example of a smart city domain
    ↪  this device belongs to."
}
```

Figure 8. ChatGPT Instructions used to Describe Data

It is important to write clear and precise instructions in a prompt to get better results. As shown in Figure 8, the instructions used ask to concisely describe the type of data being sent by the device. Since these instructions ask only request a general description of the data being sent, it was necessary to explicitly mention to avoid terms like "IoT" and "smart city", as well as references to time series.

Another effective technique was to instruct the model to adopt a persona. Without a persona, the model might occasionally describe the JSON object structure. However, when specific instructions were given, such as "take the role of a data analyst who is proficient in interpreting JSON objects and extracting meaningful insights from them," the quality of responses improved significantly.

5.2.2 Input Formatting

The context length of LLMs, which is the maximum amount of tokenized text in its input and output combined for every query, prevents feeding the entire dataset at once. Similar to summarizing a lengthy text like a book, the data can be divided into chunks. However, this method has drawbacks as it may mix contexts from different segments, potentially

impacting the results negatively. Therefore, it was decided to query each device in the dataset separately to maintain the clarity of the context.

Some events in the database are very large, with a single event containing more than 100,000 characters, which translates to over 40,000 input tokens, often exceeding the token limit of LLMs. To manage this, a JSON schema of the event object was created, and only the first item from a list of values for each required field was selected. This approach eliminated unnecessary repetition and enhanced the quality of the input.

5.2.3 Structured Prompting

When asking questions from an LLM, all prompts, schemas, and outputs are in string format. Parsing the output can be unreliable. For example, if the response is expected to be in JSON format, the output can contain hallucinations, variations in JSON key spelling, or missing values. This requires approach required a lot of additional work to validatidate the structure of the desired result.

To address this issue and enhance prompt engineering, consider using libraries such as Marvin or Instructor. These libraries enable returned prompts to be formatted as a data structure rather than just plain text. Typically, relying on LLMs to perform multiple tasks in a prompt is unreliable and error-prone. Leveraging these libraries allowed the response to be formatted as an object containing just the domain, subdomain, and description field. This structure made sure that the responses contained only relevant information.

Table 4. GPT-4 Turbo Description for an Electric Bike Dock

Domain	**Description**
Urban mobility Parking management	The device measures the number of bicycles parked at a specific location.
Transportation Bicycle sharing systems	Data on the number of bicycles docked
Urban mobility Bicycle tracking	The measurement data sent by the device focuses on counting bicycles.
Urban mobility Bicycle parking management	Monitoring bike usage in urban recreational areas through a device measuring docked bicycles.
Transportation Bicycle-sharing systems	Collects data regarding the number of bicycles docked at a given location.

Large language models are non-deterministic, meaning that for a prompt, the answer can vary significantly based on how specific the instructions were. Figure 4 illustrates examples of different answers for the same prompt. Varying responses do not impact the results of the topic model as long as the relevant keywords are present.

5.2.4 Cost of Transforming Data

LLMs are pre-trained models that have been trained on a wide range of text sources to develop a broad understanding. However, their performance can be in improved specific domains through a process called fine-tuning. This involves further training the model on a specific dataset that consists of relevant prompts and responses. By doing this, the quality of responses for a specific task can be enhanced while still retaining general language knowledge. If fine-tuning the model is not possible, a similar result can be achieved by incorporating sample user questions and answers in the prompt. However, both of these approaches come at an additional cost, as LLMs are priced per token in input and output combined.

The LLM chosen for the purpose of describing data was ChatGPT-4 Turbo as at the time of writing this paper it is number one in the LMSYS Chatbot Leaderboard [16, 17], which is a crowdsourced open platform for LLM evaluations. The number two and three models are Gemini-Pro and Claude 3 Opus, but the API for those two models is not available in Estonia.

Table 5. ChatGPT Model Pricing [18]

Model	Training	Input	Output
gpt-4-turbo-2024-04-09	-	$10.00 / 1M tokens	$30.00/ 1M tokens
fine-tuned gpt-3.5-turbo	$8.00 / 1M tokens	$3.00 / 1M tokens	$6.00 /1M tokens
gpt-3.5-turbo-0125	-	$0.50 / 1M tokens	$1.50 / 1M tokens

Depending on the size of the dataset, using ChatGPT-4 Turbo might not be the most cost-effective choice. As shown in Table 5, with current pricing, ChatGPT-4 Turbo is twenty times more expensive compared to ChatGPT-3.5 Turbo, which can perform relatively well for describing data, assuming the goal is to get a general overview of what the data represents. The cost to describe 15,000 devices in the dataset with ChatGPT-3.5 Turbo is about 3 dollars, and for ChatGPT-4 Turbo, the price would be 60 dollars. This can be considered fairly pricey for such a small dataset.

5.3 Hierarchical Topic Modeling

Smart city IoT devices have different functions and produce various types of data. To distinguish these devices based on their operational characteristics and the types of data they generate, a hierarchical topic model can be utilized. This model groups similar objects into clusters, which can be nested within larger clusters, creating a tree structure. BERTTopic allows the creation of a hierarchical topic model.

By default, the topics created consist of a list of representative words for each topic. To make the overview more easily understandable, it's possible to apply a representation

model for a BERTTopic model. This representation model allows the creation of human-readable labels for each topic by utilizing ChatGPT-4. Compared to the cost of describing every device, creating labels for topics is much cheaper, as only a couple of representative documents can be used to create a description of a topic.

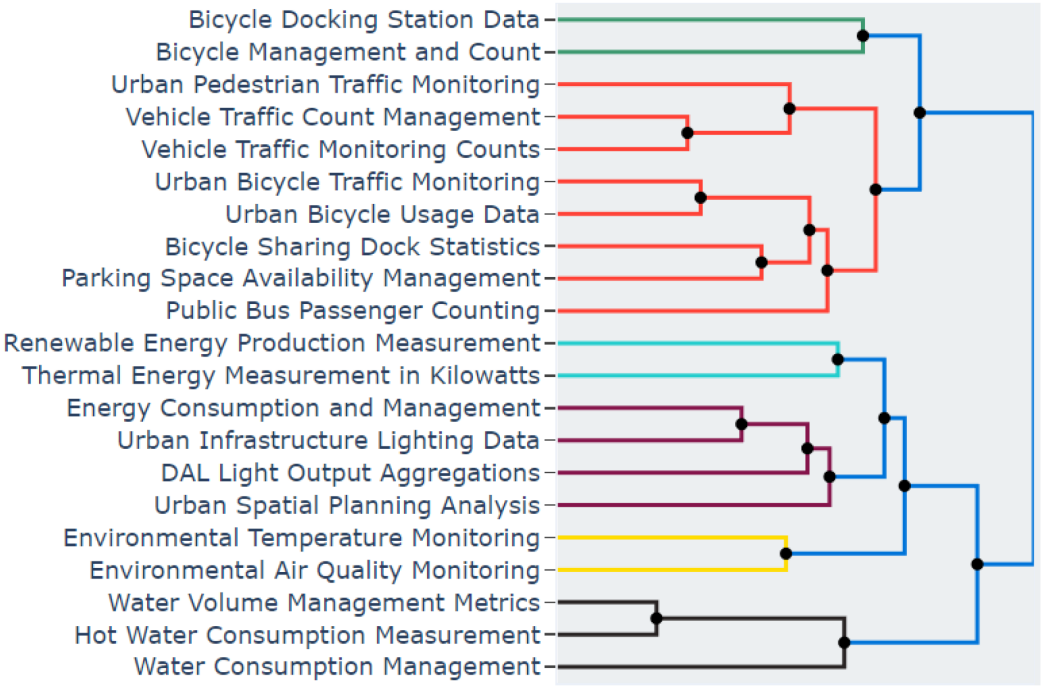

Figure 9. Hierarchical Topic Model for March 2024 Measurement Data

To create the topic model, the generated ChatGPT description was combined with suggested domains, and subdomains as input for BERTTopic. Measurement and event representations weren't used, as this would result in devices getting clustered based on similar naming or structure of data. BERTTopic was configured to automatically select the optimal number of topics and for the embedding model, OpenAI text-embedding-3-large was chosen.

As illustrated in Figure 9, it can be seen how to topic model successfully separated various types of devices into hierarchical clusters. Different colored clusters group

individual topics based on their similarity. These larger groups describe sensor data from electric bike docks, traffic counters, utility meters, street lights, and environmental sensors.

ChatGPT-4 wasn't able to completely ignore the device-specific keyword `DAL` for describing measurement data, as one of the topics is `DAL Light Output Aggregation`. Out of 7173 devices that contain `DAL` in the device or measurement name, only 167 device descriptions contained that keyword. All such devices are part of the `DAL` topic. However, the topic is clustered together with other streetlights, so the results are as desired.

The topic model does not assign topics to all input data. Approximately 20% of the input is categorized as outliers, meaning that the descriptions of these devices are not considered when creating topic descriptions and hierarchies. To minimize the number of outliers, they can be fed back into the trained model. However, this process may lead to worse results and more devices being categorized incorrectly. Consequently, the topic representations would also need to be updated with the added input.

5.4 Text Editing

In addition to utilizing ChatGPT-4 [19] to transform the model input data, it was also used to enhance the clarity and readability of the text in the thesis. The main benefit was the ability to use freewriting to compose a block of text without being concerned about grammar, spelling, or overall coherence. The model would transform the semi-structured text into a coherent paragraph by eliminating redundant phrases and presenting the main ideas more clearly. This generated paragraph could then be used as a guide for writing the actual text. The model also provided help with changing the structure of a text after a sentence or two that felt out of place were removed or provided suggestions for how that part of the text could be changed. Grammarly Generative AI [20] was also used for the same purpose, mainly using the prompt "Improve it".

6 Data Overview

In this chapter, the process of data retrieval through the Cumulocity API and its subsequent validation are detailed. Furthermore, the creation of a network graph using the D3 JavaScript library to visualize the dataset, aiding in the identification of issues with data naming standards, is described.

6.1 Requesting Cumulocity Data

In order to gain an overview of the data, the initial step was to create a program to facilitate this. Since direct access to the database was not available, all data requests had to be routed through the API. Currently, the Tartu Cumulocity database contains over 19000 devices that have collectively sent out more than 400 million measurements and 700 million events. Many devices send out over a hundred thousand data points per month.

There are more than 400 open-source projects related to Cumulocity for plugins and other applications. However, none were found that would provide a comprehensive overview of the entire dataset [21]. The official Cumulocity Python API [22] was the only open-source tool used for requesting data, which ensures that the program uses the default Cumulocity configuration and can be utilized to request data from any Cumulocity system.

Algorithm 1: Requesting Data Overview from Cumulocity

```
request device Inventory;
for every device do
    request supported fragment + series;
    if device has supported fragment + series then
        request total measurement count;
        for each month do
            request total measurement count;
            for every fragment + series do
                request count of data for fragment + series;
                for every type + fragment + series do
                    request measurement count for type + fragment + series;
                end
            end
        end
    end
    for each month do
        request total event count;
        for each event type do
            request event count for that type;
            for each event type + fragment do
                request event count for type + fragment;
            end
        end
    end
end
```

The entire Tartu Cumulocity dataset is currently 1.3 terabytes in size, making it impractical to request it over the network just to get an overview. Doing so would be slow and require a significant amount of storage. To prevent requests from timing out, the data statistics time frame has been limited to one month. This monthly overview also provides a more detailed way to detect trends in data metrics over time. The process is outlined in Algorithm 1. However, even for a month, some requests for measurement and event counts still get timed out. In these cases, the requests are retried using a smaller time frame. Requesting a smaller amount of data resolves the issue of requests timing out.

6.2 Schema Validation

It is expected that every measurement and event received has a type, but the API does not offer a way to request the types used by a device. When a request is made for an event or measurement count, the most recently recorded event or measurement within the specified time frame is also returned. This allows for the mapping of event and measurement types for subsequent requests by using previously seen data. If a device transmits only one type of data, then this is sufficient for mapping data types. However, for devices that send out multiple types of events or measurements, some types might be absent from the retrieved data. Mapping previously seen types makes the process of requesting data much faster, as the alternative would be to request different parts of the data until a new type is seen. This can be done by continuously requesting data or requesting random portions of data. However, the worst-case scenario is always that all the data within that time frame needs to be requested, making this process slow and ideally avoided. All types have been accounted for when the sum of their respective types equals the total sum.

An alternative method to map types and fragments for measurements and events would be to create a WebSocket client that connects to both measurement and event feeds. This client would continuously identify and catalog any new types and custom fragments as they are transmitted. Due to the potential volume of devices and data involved, queries for historical data can often be resource-intensive and slow. So implementing a real-time tracking system would eliminate the need for historical data queries for sensors that are actively transmitting data. However, this would come with the cost of having this application continuously running, as some events or measurements might only be sent once a month, and depending on the amount of incoming data, this application would need to be scaled as well to accommodate the increased load. So, the cost of running this application overweight the gain in provides.

Count	Fragment
11,472,347	timestamp
9,624,526	crossedPaths, label, polygons
1,847,821	insert_timestamp
775,394	count
775,133	avg_speed, max_speed, min_speed, speed_log
643,592	bottom_humidity, bottom_temperature, gyro_angle_x, gyro_angle_y, gyro_angle_z, internal_humidity, internal_temperature, light_level, road_temperature, top_humidity, top_temperature
429,096	amperage, apparent_power, frequency, power_factor, reactive_energy, reactive_power, total_kWh, voltage
422,321	active_power
398,591	active_energy
267,563	entities, header
11,821	current, location

Table 6. Unique Event Fragments for "Dal" Devices Based on 12,732,422 Event Objects

Event-specific fragments (any additional JSON keys in the data object) can be mapped in a similar way based on previously seen data. Each event can have any number of custom fragments. Therefore, the total sum of the custom fragment count is always equal to or higher than the total number of events. However, it can be difficult to validate whether all custom fragments were seen if the event representation varies across similar devices. As shown in Table 6, it is not clear whether all custom fragments were seen based on numbers alone. Due to the variability in event representation, it is not always possible to guarantee an accurate overview of all utilized fragments without requesting a week or even months' worth of data.

6.3 Dataset Visualisation

An effective way to identify problems with the current dataset is by visualizing it. Using a knowledge graph is a good choice for this as it can help to reveal the relationships between different elements within the dataset.

6.3.1 Interactive Graph Visualization

To visualize the dataset, it was decided to use the D3 JavaScript library that enables the creation of interactive and dynamic data visualizations in web browsers. D3.js is a free, open-source JavaScript library for visualizing data [23]. D3.js enables the simulation of forces such as attraction, repulsion, and gravity between nodes. Force simulation can be used to prevent node overlap and, when applied to edges, would help in creating clusters of closely related nodes. With a combination of positional forces of different strengths, the network can be made to represent nodes in a hierarchy structure as well.

The graph is interactive, meaning users can move nodes around and zoom in and out to explore different parts of the graph to get a better understanding of the data connections. Users can hover over individual nodes to see more detailed information, such as the device and measurement count related to that node. Node visibility can be adjusted as well. By clicking on a node, only the subgraph connected to the node is highlighted, remaining subgraph remains faintly visible.

6.3.2 Layered Graph Structure

The graph is organized into several layers, each representing a separate non-inclusive part of the dataset. Each layer in the network is color-coded, making it easy to distinguish between different types of information at a glance. Spatial separation between the layers helps to visually guide users through the logical progression of the data from general device types down to specific measurement units.

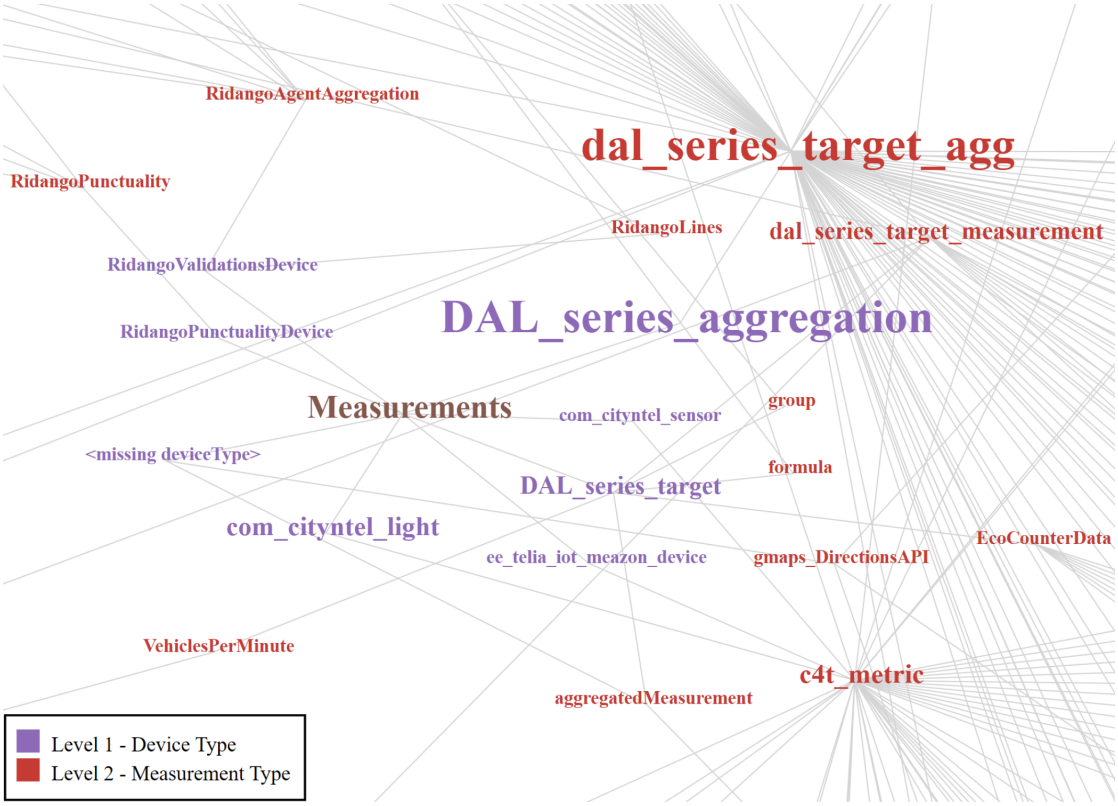

Figure 10. Measurement Graph Inner Layers

The structure mirrors the steps one would typically follow to sift through the dataset and find specific information. The innermost layer groups nodes according to the type of the device. This layer sets the context for the data exploration, showing the broadest categories first. The device type reflects its purpose and the kind of environment it operates in.

The second layer is the measurement type, which is present for every measurement and should generally represent a broad category for sent measurements. Considering the previously mentioned problems with mapping measurement types, one device shouldn't use multiple types to send out similar data. The inner two layers are in the middle as they have considerably fewer unique values compared to the outer layers, as seen from Figure 10, making the graph easier to follow.

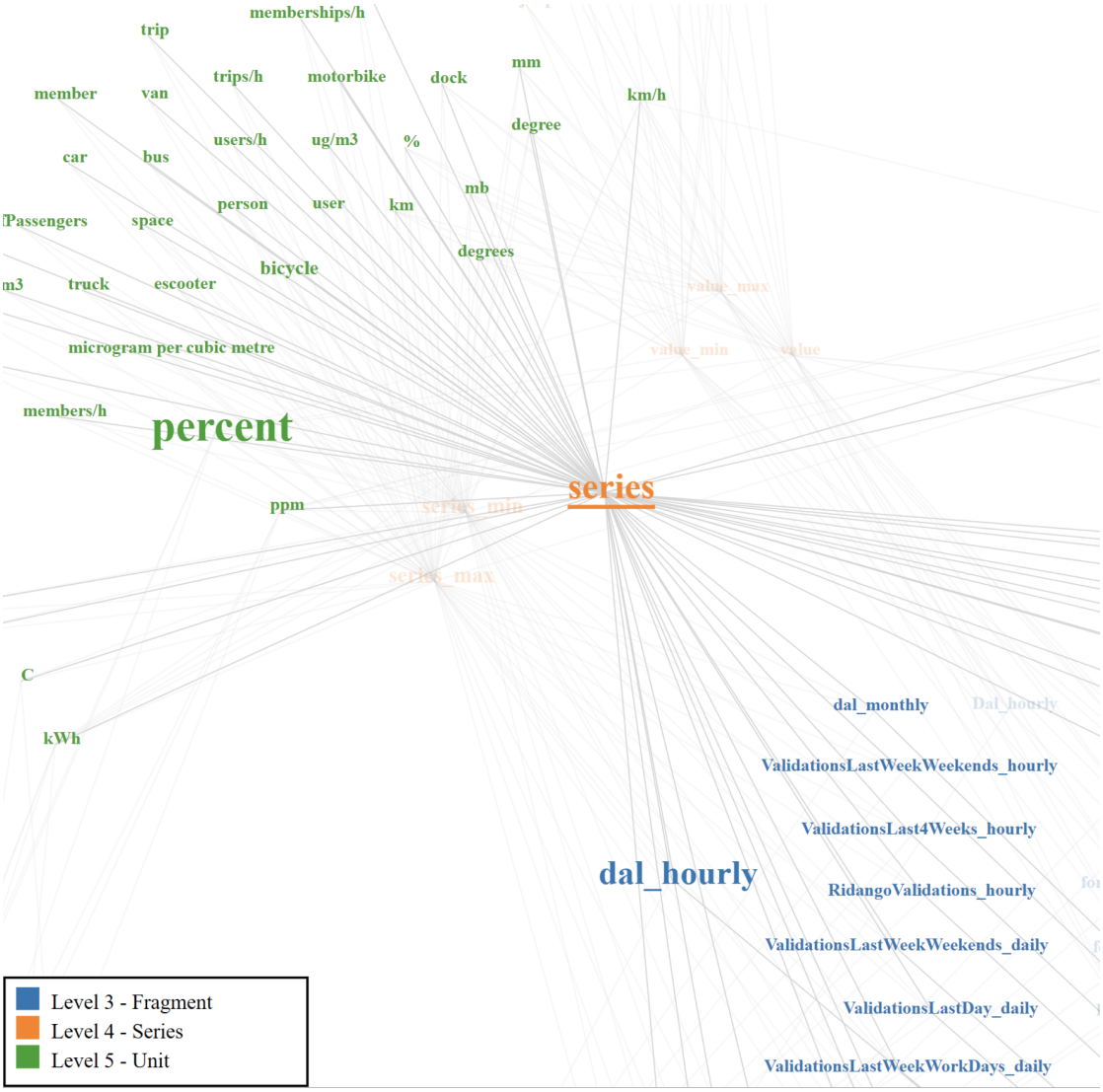

Figure 11. Measurement Graph Outer Layers

The next three layers (fragment, series, unit), illustrated in Figure 11, reflect the hierarchical measurement structure within Cumulocity, as every measurement is expected to have a value for fragment, series, and unit. The fragment and series layers provide two levels to categorize measurement data, the fragment being the broader category. Since one fragment can contain multiple series values, it makes sense to keep the fragment as the inner layer and have a series layer after that.

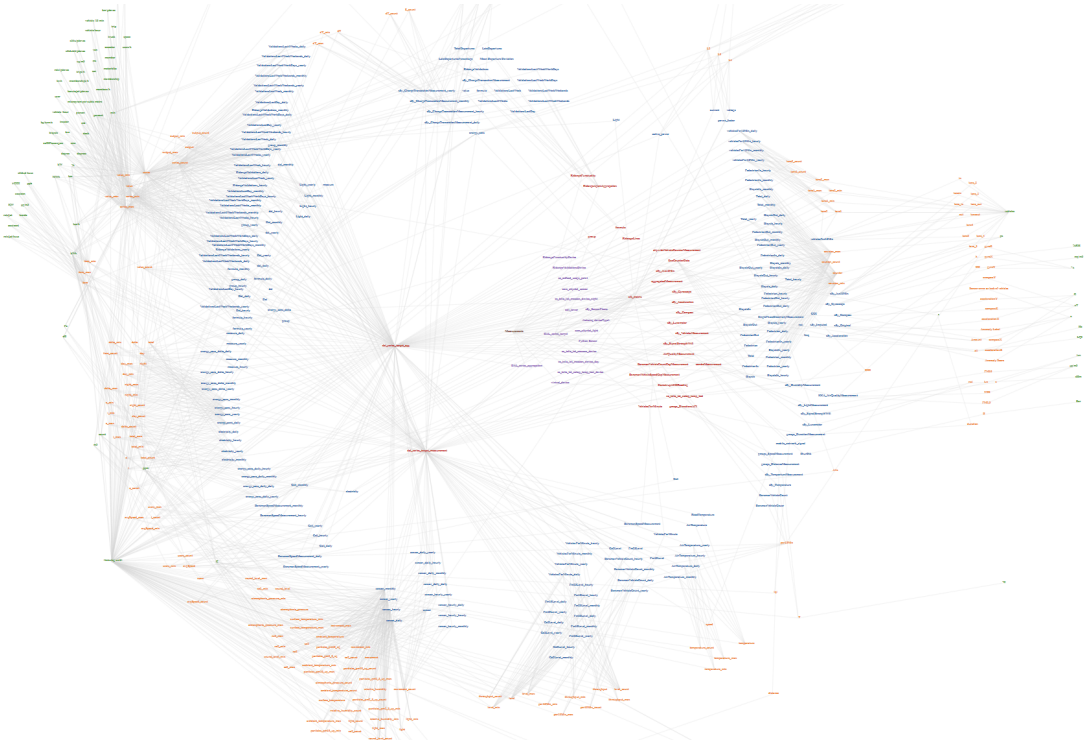

Figure 12. Measurement Graph Scale

Having a visualization of measurement structure and related features can already get complex, as seen in Figure 12. So adding many different types of semantic relations between nodes, similar to what a knowledge graph would have, would make the graph overly complex. This would take away from the main goal of the visualization, which is to see whether a similar naming and data structuring is being used. This data representation is a simplified version of a knowledge graph where labels of connections have been omitted as all relations are the same type. Visualizations for events follow the same structure but don't have series and fragment layers, as these are measurement-specific.

6.4 Added Metadata

Additional metadata was added to the visualized dataset to provide more context. This metadata includes the number of devices, the frequency of measurements associated with each node, and the total volume of data points. Larger nodes are used to depict nodes associated with many devices that generate a larger volume of data. This helps in understanding the volume of data generated by different parts of the network. As depicted in Figure 13, hovering over a node shows the combined metadata of all devices associated with it.

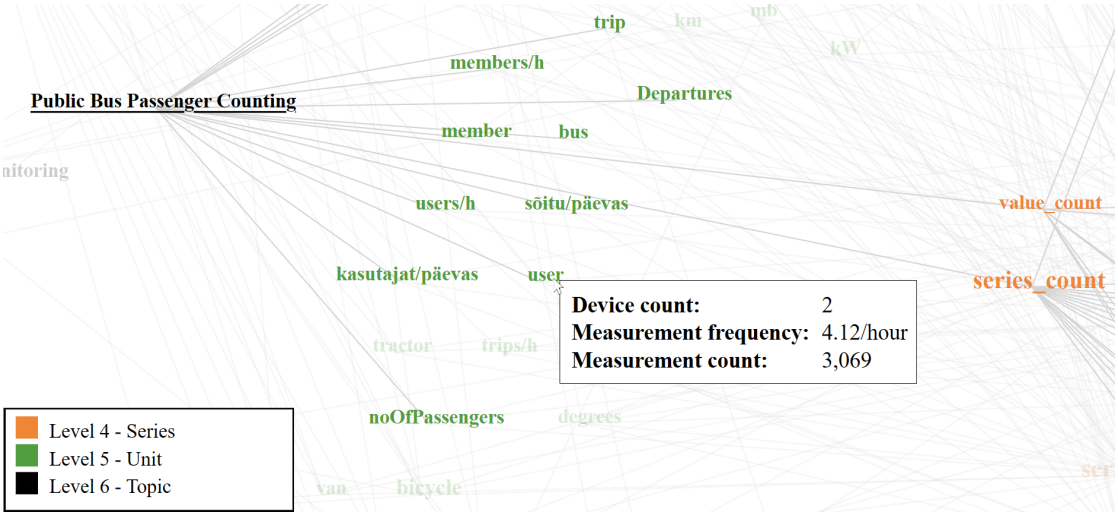

Figure 13. Measurement Graph Added Metadata

The results of the topic model were also incorporated to provide a general understanding of the data. This is helpful because some measurement categories only make sense when combined with the device name, which is not included in the visualization. The outlier topic was not included in the visualization to avoid adding unnecessary noise

To avoid confusion, the topic model was kept separate from the actual data structure and added as the last layer in the visualization. In situations where a device does not send measurements, the topic layer is linked to the series layer instead. The topic model can

be placed between any two layers of the hierarchy, as it provides a general description of the device data and is not specific to any particular layer. This functionality to change the ordering of the layers is something that can be added to the visualization in the future.

6.5 Identifying Problems

After visualizing the dataset, the problems with the organization of the dataset can be more easily seen. Measurement fragments and series naming must be clear enough to understand what kind of measurements are being sent, as both these fields are a required part of every measurement. If the fragment combined with series is sent out as `dal.series`, `measurement.item`, `formula.formula` or `group.value`, it doesn't provide any context as to what kind of data that device is sending out.

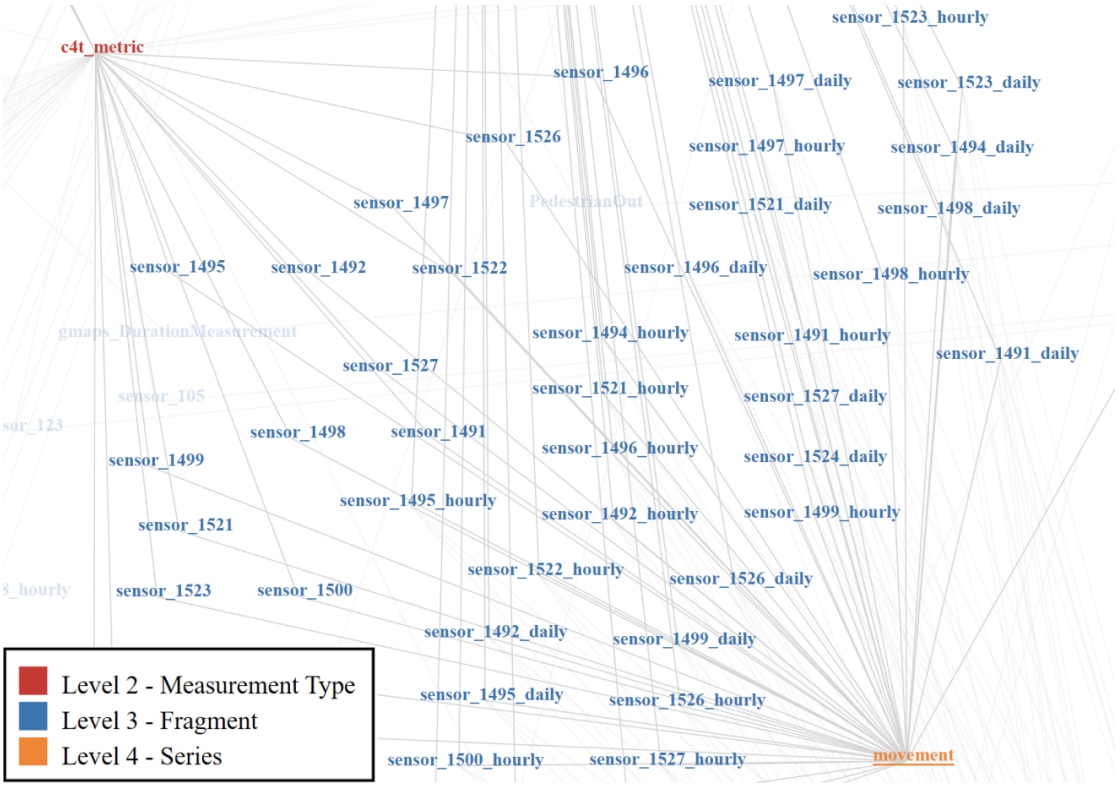

Figure 14. Measuremnt Fragment Poor Naming Practice

The sensor fragments depicted in Figure 14 have unique identifiers in their names, which makes it challenging to retrieve hourly or daily data for similar devices. To do so, one would need to map all device identities and make a new request for each sensor identifier, a cumbersome task. Measurement type and series wouldn't help this situation as these are the same for all such fragments. While the company that sends out the data might have a mapping for all such devices, this would add unnecessary complexity for anyone else looking to get an overview of the dataset.

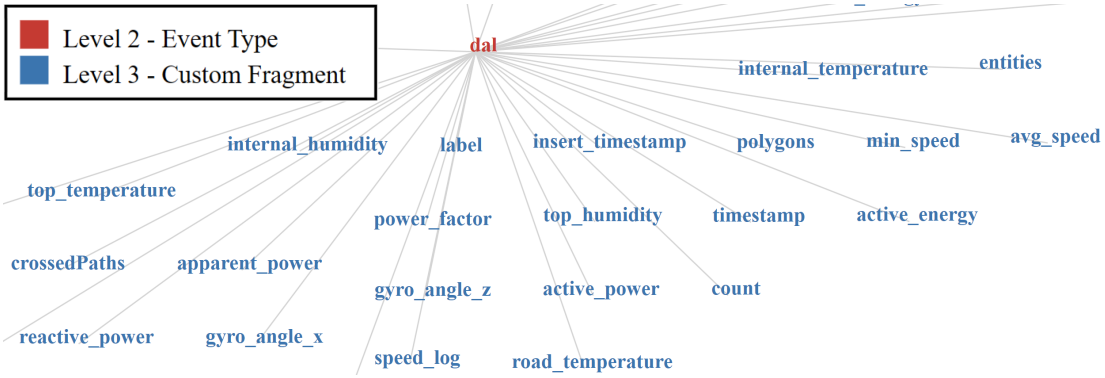

Figure 15. Measurements Sent as Events

While there aren't many different kinds of events compared to measurements, which makes it easier to get an overview of events. However, some issues can still be found. Figure 15 illustrates the use of custom fragments such as `internal_humidity`, `apparent_power`, `avg_speed`, and `entities`. As all these fields contain numeric values, it's more appropriate to send them out as measurements rather than events. When measurements are sent out as events, it may not be clear that the data is time series data. Cumulocity provides options to aggregate daily, hourly, or minute data for measurements, but this functionality isn't available for measurements sent out as events.

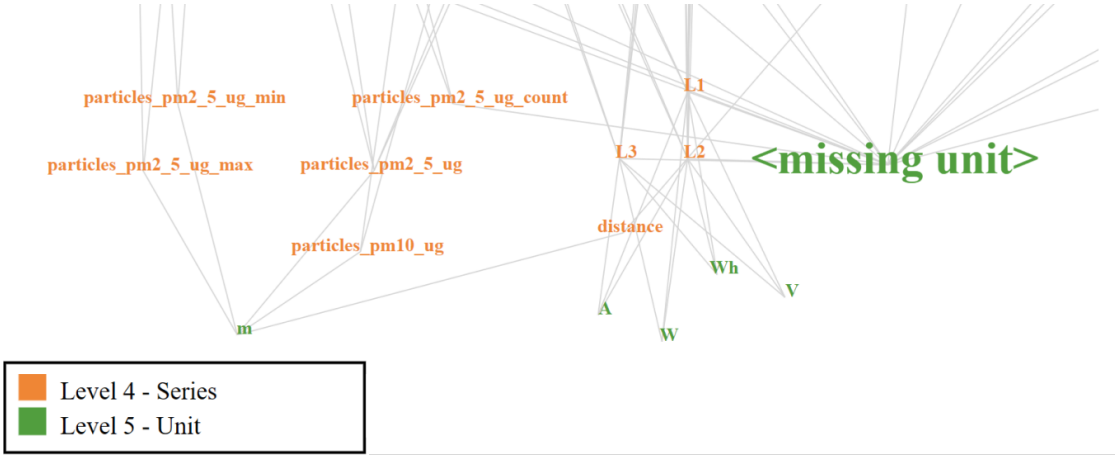

Figure 16. Inconsistent use of Units

There are inconsistencies in the use of units in the measurement data, as shown in Figure 16. While some devices leave the unit empty when sending out measurements, others specify the unit in the fragment or series name. For instance, the series value `particles_pm2_5_ug_count` includes the unit `count` in its name. It is also misleading to use the unit `m` for air quality measurements, as `m` is the standard abbreviation for meter, but in this context, it refers to particles per cubic meter.

Additionally, some devices use units but omit the unit when sending out default values for missing data. This happens when a device sends out multiple types of measurements at once, and one of the measurements is configured to be sent out less frequently than the others. Instead of not sending out the missing measurement, it is replaced with a default value.

6.6 Insights into the Dataset

The issues observed in the graph visualization arise from underlying problems within the dataset itself. Inconsistent categorization and a mix of different naming conventions are the primary factors contributing to the complex and challenging visualization. Ad-

dressing and fixing these problems can greatly enhance the clarity and usefulness of the visualization. Using descriptive names would also help the topic model more easily distinguish various kinds of data.

Consistent categorization is beneficial for visualizing data as it provides a clear overview of the types of data being collected. This makes it easier to assess how similar kinds of data should be combined to gain insights into recent data trends. Additionally, when integrating new systems or devices with Cumulocity, visualization serves as a guide on how the newly incoming data should be named and categorized to align well with the existing data.

To prevent similar issues in the future, it is advisable to conduct regular audits on the dataset to ensure consistent and accurate categorization. Additionally, establishing comprehensive documentation and utilizing automated tools to detect and flag inconsistencies in naming conventions is recommended.

6.7 The Need to Migrate Data

The government of Tartu has identified the need to transition data from Cumulocity cloud storage. This need coincides with the goals of this thesis, as the results can be directly utilized to support the city's data migration efforts. The thesis provides a detailed assessment of the volume of data, including measurements and events, and the diverse data schemas employed by various devices. Additionally, insights gained from network graph visualizations can highlight issues with the current naming conventions and help evaluate the necessity of migrating all data, especially considering that much of the measurement data is redundantly captured in daily and hourly aggregates.

7 Conclusion

This thesis focused on the challenge of interpreting complex data collected from various sensors in smart city environments. The study revolved around the dataset provided by the Tartu Cumulocity IoT platform. For this purpose, a program was developed to query and map out the data volume and schema used by various devices within the Cumulocity dataset. This functionality is not readily available in Cumulocity itself. However, due to the limitations of the API, the program can only guarantee a complete schema mapping for measurements, as complex event object representation cannot be directly requested.

A hierarchical topic model using BERTTopic modeling techniques was used to analyze the different types of smart city-related data present in the dataset. However, the initial results obtained by using data schema representation as input were unsatisfactory. Consequently, large language models (LLMs) were used to transform device descriptions, which significantly improved the quality of the input data for topic modeling. This demonstrated the usefulness of LLMs in comprehending complex object representations.

Moreover, to provide an insightful perspective on data standardization practices within Cumulocity, the dataset was visualized using a simplified knowledge graph built with the D3 JavaScript library. This visualization revealed inconsistencies and categorization issues in the current dataset, thereby highlighting the need for improved standardization to enhance data analysis.

References

[1] Software AG. Cumulocity IoT. https://www.cumulocity.com. (15.05.2024).

[2] Software AG. Cumulocity Domain Model. https://cumulocity.com/guides/concepts/domain-model. (15.05.2024).

[3] Software AG. Cumulocity API. https://cumulocity.com/api/core/10.18.0. (15.05.2024).

[4] Rui José and Helena Rodrigues. *A Review on Key Innovation Challenges for Smart City Initiatives*, volume 7, pages 141–162. 2024. doi: 10.3390/smartcities7010006.

[5] Ben Ramdani and Peter Kawalek. *Innovation and Smart Cities Research: A Review and Future Directions*, pages 1–16. 10 2023. ISBN 978-3-031-35663-6. doi: 10.1007/978-3-031-35664-3_1.

[6] RIA. Estonian Open Data Portal. https://avaandmed.eesti.ee. (15.05.2024).

[7] Software AG. Cumulocity IoT Sensor Library. https://cumulocity.com/guides/reference/sensor-library. (15.05.2024).

[8] Riccardo Tommasini, Filip Illievski, and Thilini Wijesiriwardene. *IMKG: The Internet Meme Knowledge Graph*, pages 354–371. May 2023. ISBN 978-3-031-33454-2. doi: 10.1007/978-3-031-33455-9_21.

[9] Alessandro Negro. *Graph-Powered Machine Learning*. Manning Publications Co, August 2021. ISBN 9781617295645.

[10] Graphaware. Hume - Mission-Critical Graph Analytics. https://graphaware.com/products/hume. (15.05.2024).

[11] Alessandro Negro, Vlastimil Kus, Giuseppe Futia, and Fabio Montagna. *Knowledge Graphs Applied MEAP Edition Version 2*. Manning Publications Co, June 2022. ISBN 9781633439894.

[12] David Blei, Andrew Ng, and Michael Jordan. Latent dirichlet allocation. volume 3, pages 601–608, 01 2001.

[13] Anestis Kousis and Christos Tjortjis. Investigating the key aspects of a smart city through topic modeling and thematic analysis. *Future Internet*, 16:3, 12 2023. doi: 10.3390/fi16010003.

[14] Maarten Grootendorst. BERTopic: Neural topic modeling with a class-based TF-IDF procedure. *arXiv preprint arXiv:2203.05794*, 2022.

[15] Maarten Grootendorst. BERTopic Algorithm. `https://maartengr.github.io/BERTopic/algorithm/algorithm.html`. (15.05.2024).

[16] Wei-Lin Chiang, Lianmin Zheng, Ying Sheng, Anastasios Nikolas Angelopoulos, Tianle Li, Dacheng Li, Hao Zhang, Banghua Zhu, Michael Jordan, Joseph E. Gonzalez, and Ion Stoica. Chatbot arena: An open platform for evaluating llms by human preference, 2024.

[17] LMSYS Chatbot Arena Leaderboard. `https://chat.lmsys.org/?leaderboard`. (15.05.2024).

[18] OpenAI Model Pricing. `https://openai.com/pricing`. (12.05.2024).

[19] OpenAI (2023). ChatGPT-4 (29.04.2024). `https://chat.openai.com`.

[20] Grammarly (2023). Grammarly Generative AI. `https://www.grammarly.com/ai`.

[21] SoftwareAG. Cumulocity IoT Open-Source Repository Overview. GitHub repository. https://github.com/SoftwareAG/cumulocity-os-repo-overview. (15.05.2024).

[22] Software AG. Cumulocity Python API. GitHub repository. https://github.com/SoftwareAG/cumulocity-python-api. (15.05.2024).

[23] D3: Data-Driven Documents. GitHub repository. https://github.com/d3/d3. (15.05.2024).

Appendix

I. Code Repository

The code for requesting data schema from Cumulocity, creating the topic model, and visualizing the dataset is accessible in the GitHub repository. Note that requesting data from Cumulocity requires an account with sufficient permissions while transforming the dataset requires an OpenAI API key and sufficient funds on the account. The links to the visualization can be found in the repository readme file.

URL: `https://github.com/kasparkadalipp/C8y-Data-Overview`

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public

II, **Kaspar Kadalipp**,

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Knowledge Graphs for Cataloging and Making Sense of Smart City Data,

   supervised by **Pelle Jakovits**.

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Kaspar Kadalipp
15.05.2024