DiSIF: Distributed Semantic Information Fusion Framework for Smart City Applications

Ramin Rezvani-KhorashadiZadeh, Mohsen Kahani*

Department of Computer Engineering, Ferdowsi University of Mashhad, Mashhad, Iran

Abstract

The advent of the Internet of Things (IoT) has accelerated the development of Digital Twins of Cities—virtual replicas of urban environments that enable real-time monitoring, analysis, and decision-making by integrating heterogeneous and high-velocity data streams from diverse sources. Managing these massive, heterogeneous data streams presents critical challenges including data fusion, scalability, privacy, and timely query execution. In this paper, we propose DiSIF (Distributed Semantic Information Fusion Framework), a novel architecture tailored for the Digital Twin of a City paradigm. DiSIF leverages semantic data models and RDF stream processing within a distributed semantic JDL fusion framework deployed across edge, fog, and cloud layers. This multi-layered design enables localized low-level data processing at the edge, significantly reducing raw data transmission and enhancing privacy. Horizontal fusion integrates heterogeneous sensor data to resolve semantic diversity, while vertical fusion aggregates homogeneous data to optimize bandwidth and computational resources. DiSIF also supports parallel execution of complex and dependent queries, substantially improving response times and resource utilization compared to centralized fusion models. Extensive evaluations in realistic urban scenarios demonstrate that DiSIF significantly enhances network efficiency, query performance, and scalability, making it a robust and scalable solution for implementing Digital Twins of Cities and advancing smart urban governance.

Keywords: Digital Twin of a City, semantic JDL, data fusion, fog computing, RDF stream processing, distributed architecture.

1. Introduction

Data fusion is a critical process in Digital Twins of Cities, where multiple data sources from various applications need to be integrated to improve decision-making and extract valuable insights. Digital Twins of Cities represent virtual replicas of urban environments that continuously ingest and analyze heterogeneous data streams to provide real-time situational awareness and predictive capabilities. By combining data from multiple sensors, the limitations of individual sensors, such as range or errors, can be overcome, resulting in enhanced system reliability and broader situational awareness. Data fusion improves system stability, increases accuracy, reduces uncertainty, and lowers costs, all of which are essential for the complex and dynamic nature of Digital Twin environments.

However, a major challenge in Digital Twins of Cities is the heterogeneity of data sources. Data differences in syntax, structure (schema), and meaning (semantics) can lead to issues during fusion, creating semantic conflicts and reducing system coherence. To address these challenges, a conceptual model that provides a formal, common understanding of the target domain is essential for successful data fusion in Digital Twin environments.

The development and real-time maintenance of Digital Twins of Cities face substantial data-related challenges due to the massive, high-velocity, and heterogeneous nature of IoT sensor data streams. The immense data volume, such as traffic and weather streams producing up to 10,000 RDF triples per second, places significant computational demands on edge, fog, and cloud layers, straining resource scalability. The high velocity of these real-time streams requires rapid processing to ensure timely updates, essential for critical applications like traffic signal optimization. Moreover, data heterogeneity—stemming from diverse formats, schemas, and semantics across sources like traffic sensors and weather APIs—complicates integration, often resulting in semantic conflicts that undermine the accuracy and coherence of the Digital Twin. The DiSIF framework mitigates these challenges through a distributed architecture and Semantic Web technologies, including RDF and SPARQL, enabling efficient, scalable, and privacy-preserving data fusion to support real-time urban Digital Twin operations.

Centralized data fusion models are inherently inefficient for large-scale Digital Twins of Cities due to several critical challenges. The massive data volumes generated by urban IoT sensors, such as traffic and weather streams, lead to network inefficiencies as all data must be transmitted to a single processing node, causing bottlenecks and increased latency. Privacy concerns arise from the transmission of raw, sensitive data, such as vehicle GPS coordinates, to centralized servers, risking unauthorized access. Additionally, the high query execution times in centralized systems, particularly for complex semantic queries on large-scale RDF datasets, hinder real-time updates essential for applications like traffic management. The DiSIF framework

*Corresponding author

Email addresses: ra.rezvanikhorashadizadeh@mail.um.ac.ir (Ramin Rezvani-KhorashadiZadeh), kahani@um.ac.ir (Mohsen Kahani)

addresses these limitations through a distributed architecture, leveraging edge, fog, and cloud layers alongside Semantic Web technologies like RDF and SPARQL to enable scalable, privacy-preserving, and low-latency data fusion for efficient urban Digital Twin operations.

To overcome the challenges of data volume, velocity, and heterogeneity in Digital Twins of Cities, the DiSIF framework introduces a distributed three-layer architecture comprising edge, fog, and cloud layers. This architecture enhances efficiency and scalability by processing high-volume, real-time IoT data streams locally at the edge layer, reducing network latency and preserving privacy. The fog layer handles complex semantic queries and data fusion, leveraging RDF and SPARQL to integrate heterogeneous data, while the cloud layer supports macro-level decision-making and historical analysis. By distributing computational tasks across these layers and employing parallel query execution, DiSIF ensures scalable, low-latency, and privacy-preserving data fusion, meeting the demanding requirements of real-time urban Digital Twin applications.

The JDL fusion model is a widely recognized model for data fusion, with its semantic version [31] enabling the integration of heterogeneous data using Semantic Web technology such as RDF stream processing. This model can be implemented using centralized, distributed, or hybrid architectures. However, in Digital Twins of Cities, the traditional centralized semantic JDL model, where data is fused at a single node, faces challenges due to the vast data volume and the need for multi-layered decision-making. This centralized approach is inefficient for large-scale data fusion, leading to increased response times, network inefficiencies, and costly decision-making processes, which are critical bottlenecks in real-time Digital Twin applications.

To address these limitations, the DiSIF framework introduces a three-layer architecture comprising edge, fog, and cloud layers. The edge layer handles time-sensitive decisions close to data sources, while the fog and cloud layers process more complex and macro-level decisions. This distributed architecture aligns with the layered nature of Digital Twins of Cities, enabling scalable, privacy-preserving, and efficient data fusion and decision-making.

The framework enhances the distributed semantic JDL model by employing two types of fusion: **Vertical fusion**: Scales up data without changing its concept (e.g., combining traffic data from streets to generate traffic flow for an area). This reduces query execution time by distributing sub-queries across multiple nodes. **Horizontal fusion**: Changes the concept of the data by fusing streams from different domains (e.g., vehicle movement and congestion data to create a "traffic" concept). This also optimizes query execution by distributing dependent sub-queries across nodes. By incorporating these fusion strategies, the DiSIF framework significantly reduces execution time and improves resource efficiency compared to centralized approaches, making it more suitable for the demanding requirements of Digital Twins of Cities.

Additional strengths of the DiSIF framework include:

- **Execution of Independent Queries in Parallel**: Independent queries can be decomposed into sub-queries executed concurrently across distributed nodes, with results combined at higher layers. This parallelism significantly optimizes query execution time, a crucial factor for real-time Digital Twin operations.
- **Execution of Complex and Dependent Queries**: DiSIF supports dynamic execution of complex queries introduced at runtime, relying on previously executed sub-queries. This flexibility is essential for adaptive and evolving Digital Twin scenarios.
- **Reduction of Network Load**: By processing data locally and transmitting only processed results, DiSIF reduces network bandwidth utilization, addressing one of the main challenges in large-scale Digital Twin data management.
- **Enhancement of Data Privacy**: Local data processing avoids transmission of raw data, preserving privacy—a critical concern in urban Digital Twin deployments.

As stated earlier, the primary goal of this article is to present a distributed version of the semantic JDL fusion model tailored for Digital Twins of Cities. While various advancements have been made in different aspects of JDL fusion models, a fully distributed approach optimized for the scale, complexity, and real-time requirements of Digital Twins has not been explored until now. The innovations proposed in this article can be summarized as follows:

1. **Distributed Semantic JDL Fusion Model Across Three Layers (Edge, Fog, and Cloud)** This model supports multi-layered decision-making aligned with the hierarchical architecture of Digital Twins of Cities, where micro and small decisions at lower layers are fused to form macro-level decisions at higher layers. The DiSIF framework enables high-speed, parallel execution of macro decision-making processes across different layers, outperforming traditional centralized JDL fusion models. To enhance processing speed and manage complexity, DiSIF separates fusion operations from decision-making tasks. Unlike centralized models where all fusion and outputs share a common bus platform, DiSIF isolates fusion and decision-making at each layer, enabling parallel and independent fusion operations that better suit the distributed nature of Digital Twins.
2. **Proposing Two Fusion Models: Horizontal Fusion (Data Concept Change) and Vertical Fusion (Data Scale Change)** Previous fusion models typically operated on data within a single concept domain. In contrast, DiSIF supports fusion across multiple concepts relevant to Digital Twins of Cities, such as integrating traffic and weather data to generate enriched urban context or new composite concepts. Horizontal fusion thus enables semantic enrichment and concept evolution. Vertical fusion aggregates data from similar concepts but at different scales—for example, fusing street-level traffic data into area-wide traffic flow—allowing for more comprehensive and precise macro-level decision-making with a broader urban perspective.

3. **Introducing a Distributed Query Execution Model** DiSIF presents an optimized, distributed approach for executing queries within the three-layer architecture. Independent queries are decomposed and executed across layers, with worker nodes in upper layers and master nodes in lower layers collaborating efficiently. Dependent queries, which rely on prior query results, are executed within layers and between worker and master nodes, supporting dynamic and complex query workflows typical in Digital Twin scenarios. This distributed query execution model significantly reduces query execution time, minimizes memory consumption, and optimizes network bandwidth usage, all critical for real-time Digital Twin operations.

It is important to note that while this article does not introduce innovations specifically in the cloud domain, it leverages the cloud's computational power for complex fusion tasks and macro-level decision-making within the Digital Twin framework. Decisions generated at the fog layer are forwarded to the cloud node for final aggregation and strategic urban management. The structure of the article is as follows: Section 2 reviews related works on data fusion models, Semantic Web approaches in smart cities, fog computing architectures, and RDF stream processing techniques. Section 3 details the DiSIF framework from the perspectives of the semantic JDL model and the proposed fusion , with a layer-by-layer discussion. Section 4 presents comprehensive evaluations and analyzes the results. Finally, Section 5 concludes the article. Additionally, the queries used in this study are provided in the Appendix A.

2. Related works

This section reviews recent works for data fusion and analyzes various data fusion models. It also discusses studies that have utilized Semantic Web technologies in smart cities, explores fog computing architectures in smart city contexts, and reviews approaches to RDF stream processing.

2.1. Data fusion methods

The process of data and information fusion involves combining inputs to create richer information than what can be obtained from each input separately.

One of the common models in the field of fusion is the multi-level JDL model [23], which with its 5 different levels covering from raw data processing to decision-making, has contributed to the comprehensiveness and popularity of this model in the fusion domain.

The semantic JDL model [31] combines the basic JDL model with Semantic Web technologies. The JDL semantic model in Figure 1 is structured into several levels, each addressing different aspects of data processing. At Level 0, the focus is on preprocessing various data sources, resulting in cleaned data that is forwarded for object refinement without introducing any semantic elements. Level 1 involves transforming objects and their attributes into standard RDF format for storage. While this level includes tasks such as object identification, these processes

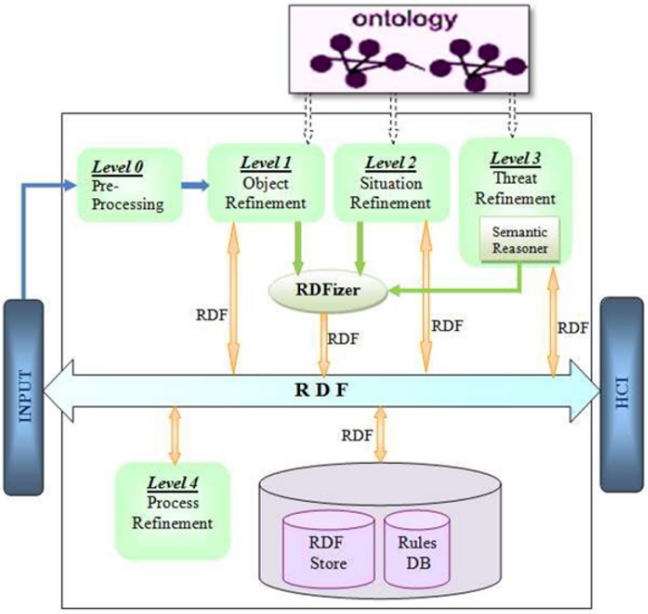

Figure 1: The semantic JDL model [31]

rely on mathematical algorithms and image processing rather than semantic definitions. To effectively structure the data, predefined ontologies are necessary, allowing for the integration of attributes into the RDF format using an RDFizer, with the data stored in an RDF-Store database. In Level 2, the model uses prior knowledge and environmental information alongside RDF data from Level 1 to define the situation of objects and their interrelationships. New relationships and previously unknown attributes are identified through inference, leading to updates in the RDF data that reflect any changes in information. Level 3 emphasizes the evaluation of the current situation and the prediction of potential threats and vulnerabilities. A semantic reasoner plays a crucial role at this level, employing inference techniques to assess the situation and identify solutions and opportunities related to threats. The results of this analysis are converted into RDF format for storage. Finally, Level 4 focuses on monitoring the system's performance and resource allocation. An expert evaluates the outputs from Level 3 to make informed decisions that enhance the overall efficiency of the system. The JDL semantic model operates with multiple databases, including one for RDF information and ontologies and another for rules. Proper database management principles must be upheld to ensure compatibility, prompt updates, and efficient data handling. Specific details regarding inputs and outputs may vary based on the domain of application.

One of the notable features of JDL, which makes it suitable for distributed architecture, is its ability to effectively separate tasks. One of the advantages of the JDL fusion model over other fusion models is the separation of fusion stages into completely distinct parts. This allows for the distribution of different components of the JDL model across various systems, enabling distributed fusion operations. Therefore, in this article, the JDL fusion model is utilized to present its distributed version for

smart traffic application.

2.2. Semantic Web in smart city

Architectures of smart cities should combine data received from sensors in a manner that is readable by machines and publishable. To add meaning to the raw data generated by sensors and enhance data interaction and integration, Semantic Web technology has been employed. Semantic Web adds new information to the architectures of Internet of Things (IoT) for sensor fusion.

In [35], the LSM architecture is introduced to achieve data interaction through the Semantic Web by integrating sensor data. It provides a user interface for publishing, annotating, and querying sensor data, utilizing the SSN ontology for describing sensor data and streams. The architecture links time-dependent data with external resources through semantic annotations, standardizes diverse data formats, and incorporates contextual knowledge from sources like DBpedia and GeoNames to enhance advanced querying. This approach does not address query decomposition and distributed query execution and does not consider optimizing query execution. Additionally, it considers the execution of CQELS queries centrally and does not consider the cost of sending data to the central node.

In [3], the SDFF framework is proposed to integrate data from heterogeneous sensors using Semantic Web technologies to resolve inconsistencies, such as differences in measurement units. It consists of layers for raw data collection, storage (separate repositories for raw and semantic data), semantic annotation, conflict resolution, and data dissemination. The framework ensures accurate data fusion and comparison, offering a comprehensive solution for managing and harmonizing diverse sensor data. This article does not discuss topics such as distributed processing and query execution, query execution optimization, processing of large streaming data, and prevention of sending data to only one central node (centralized approach).

In [38], a framework is introduced to combine and aggregate heterogeneous data streams from sensors, transforming them into feature streams to reduce data volume and increase efficiency. In [33], sensor data is converted into RDF based on domain ontologies, focusing on reducing transmitted data and optimizing bandwidth, but it lacks discussion on query processing and execution optimization. In [14], the Sense2Web framework enhances data integration by semantically representing sensor features and linking them to external resources, facilitating seamless aggregation and system interaction.

In [22], a novel architecture is introduced for aggregating heterogeneous sensor networks by converting sensor measurements into semantic data and using ontologies for enhanced data aggregation. However, it does not cover a distributed three-layer architecture for data collection, query processing, or bandwidth management. In [43], another architecture integrates heterogeneous IoT data with a data aggregation layer to unify and improve data quality. Although it enhances data fusion at a central node, it lacks an optimized query processing model and a distributed approach that could improve system performance.

The SIGHTED architecture [30] collects and disseminates sensor data but faces challenges due to its centralized approach. Annotated data is stored and queried later, which increases query response times. Drawbacks include lack of query optimization, inability to handle large data streams, and reliance on sending all data to a central node. In [45], a semantic framework with three layers—data collection, processing, and a semantic layer—ensures data consistency and annotation through ontologies. However, it lacks distributed processing and centralizes data, leading to inefficient query execution and no optimization for large data transmission. In [8], a framework utilizing Streaming Virtual Knowledge Graphs integrates semantic data streams using OBDA. While effective for data integration, generating ontologies from RDBMS databases is time-consuming and inefficient for decentralized environments like smart cities, lacking efficient query decomposition and execution capabilities.

In [46], the study addresses semantic interoperability challenges in smart cities where diverse IoT solutions generate large data volumes exchanged via APIs. It highlights the role of ontologies and shared vocabularies to enhance environmental sensing and wellness monitoring. By using sensor-agnostic APIs and ontology modules for mobile crowd-sensing, the framework improves data integration, scalability, and real-time responsiveness in IoT applications. Privacy concerns in smart cities are addressed by the 'Ontology-Based Privacy-Preserving' (OBPP) framework [19], which uses ontologies and semantic reasoning to tackle heterogeneity, privacy, and service provision. Additionally, Semantic Web technologies play a key role in Agriculture 5.0 by improving data interoperability, accessibility, and real-time operations in the agricultural sector [16].

2.3. Fog Computing architectures

Cloud computing offers extensive resources for handling complex tasks in smart cities [18], but it has limitations such as high latency, lack of contextual awareness, and inadequate mobility support, which impede real-time processing. Edge computing addresses these issues by extending cloud capabilities to the network edge, providing localized processing and storage to reduce latency and improve bandwidth efficiency, making it ideal for real-time smart city services [2], [36]. Additionally, cloud and fog computing are explored to bring cloud resources closer to the edge, enhancing the performance of smart city systems [1].

Perera [34] explores real-world fog computing applications in agriculture, healthcare, and transportation but does not cover Semantic Web-based approaches. In [24], fog and edge computing are compared with cloud computing in smart environments, focusing on privacy, energy consumption, and challenges, but without integrating Semantic Web solutions. Shi [40] highlights the benefits and challenges of edge computing, including privacy and service optimization, through case studies, but lacks Semantic Web integration. Recent research on fog computing and the Internet of Everything (IoE) [6] addresses latency reduction and resource constraints, emphasizing scalability and real-time capabilities but provides limited detail on smart city applications. In [21], a three-layer architecture called Rainbow uses intelligent agents in smart city IoT systems but omits Semantic Web technology for data fusion. In [41], a tiered-edge architecture introduces semantic stream processing for workload

distribution but lacks ontology-based query decomposition and efficient sub-query handling.

FogBus [42] is a framework for cloud-fog integration, improving performance by activating cloud resources during overload, but it lacks semantic data processing and query decomposition. The "Analytics Everywhere" architecture [12] uses edge, fog, and cloud layers for smart parking analytics but does not optimize user requests or use RDF for data fusion. A four-layer fog architecture [5] focuses on context awareness and low latency but lacks Semantic Web and data fusion models. Dastjerdi's five-layer architecture [13] misses a distinct fog layer and fails to address semantic issues. In [32], a collaborative IoT architecture using agent-oriented algorithms and CEP does not support semantic or heterogeneous data processing. A CR edge processing platform [9] improves cloud efficiency but lacks high-level query translation and load balancing strategies. Recent studies [44] highlight edge computing's role in enhancing data quality through semantic enrichment and event processing in smart cities. To address data integrity challenges in fog computing, [39] introduces a verification protocol using SIS and identity-based signatures, improving security and efficiency.

2.4. RDF stream Processing appoaches

Real-time processing of large data streams has led to the development of RDF stream processing (RSP) models and continuous querying languages aimed at addressing the challenge of heterogeneous data. Systems such as EP-SPARQL [4], SPARKWAVE [26], and INSTANS [37] utilize temporal operators, while others like C-SPARQL [7] and CQELS [27] rely on sliding windows for continuous query execution.

RSP system implementations are generally categorized into distributed and centralized models. Distributed approaches, such as DRSS [17], built on the Apache Storm platform, and CQELS Cloud [28], leverage frameworks like Spark Streaming, Flink, and Storm for parallel stream processing. While these models enhance scalability and parallel execution, they often introduce complexities in implementation, upgrading, and usage. Centralized models, including C-SPARQL [7], SparqlStream [10], and SPARKWAVE [26], struggle with processing capacity and exhibit limitations in scalability, concurrent query handling, and collaboration.

MAS4MEAN [29] addresses the limitations of centralized models by adopting a multi-agent approach that parallelizes query processing through multiple instances of the C-SPARQL engine. Despite its ability to manage large event volumes, MAS4MEAN faces challenges in accelerating complex queries, performing local query execution, and avoiding redundant sub-query execution, leading to bandwidth inefficiency and increased query times as data and query complexity grow.

While continuous query operators for SPARQL have been developed to address stream heterogeneity, challenges related to parallelization and scalability persist. Methods such as DIONYSUS [20] and CQELS Cloud [28] focus on distributing and processing large-scale RDF streams in parallel. Efficient partitioning of queries and data across nodes, with minimal data exchange, remains essential for optimizing the processing of RDF data streams at scale.

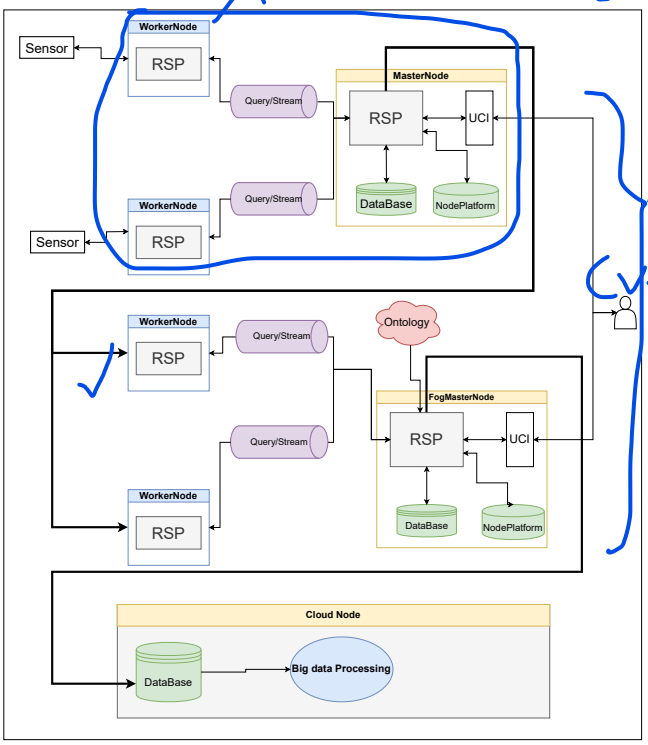

Figure 2: DiSIF framework

The article [17] introduces a scalable distributed approach for RDF stream processing by leveraging query rewriting, partitioning, and RDF graph partitioning to minimize inter-node data exchange. However, it lacks a task assignment strategy and does not implement a master-worker framework, leaving some execution details unclear.

The Waves method [25] utilizes the Apache Storm framework to distribute C-SPARQL queries across nodes, effectively handling large data volumes. However, it does not incorporate query decomposition, leading to redundant executions and inefficient query performance.

StreamQR [11] rewrites C-SPARQL queries into a Union of Conjunctive Queries (UCQ) based on ontology, injecting domain knowledge into the query. While this allows parallel execution, it can create large queries with multiple unions, increasing execution costs without optimizing time window lengths or conditions.

Table 1 outlines various frameworks and their capabilities, including query decomposition, prevention of redundant query execution, and whether they employ a layered architecture. The DIVIDE platform [15] dynamically adapts IoT stream queries based on real-time contexts using Semantic Web technologies but focuses mainly on dynamic query adaptation, leaving some performance aspects dependent on network conditions.

3. DiSIF: Distributed Semantic Information Fusion framework

In this section, we introduce the distributed version of the semantic JDL model within the framework of a three-layer archi-

Table 1:
Summary of related works

Method	Query Decomposition	Duplicate Query Execute	Handle Large Data Stream	Query Processing	DataType	Layered Architecture	Query Optimization
Zafeiropoulos et al. 2008 [45]	✗	-	✗	C	RDF	✓	✗
Patni et al. 2011 [33]	✗	-	-	C	RDF	✗	✗
De et al. 2012 [14]	✗	-	-	C	RDF	✗	✗
Phuoc et al. 2012 [35]	✗	-	-	C	RDF	-	✗
Gyrard et al. 2013 [22]	-	-	-	C	RDF	✗	-
Nagib et al. 2016 [30]	✗	-	✗	C	RDF	✗	✗
Dastjerdi et al. 2016 [13]	-	-	-	C	Non-RDF	✓	-
Giordano et al. 2016 [21]	-	-	-	-	Non-RDF	✓	-
Khrouf et al. 2016 [25]	✗	✓	✓	D	RDF	✗	✗
Calbimonte et al. 2016 [11]	Syntactically	✓	✓	C	RDF	✗	✗
Wang et al. 2017 [43]	✗	-	-	C	RDF	✓	✗
Arkian et al. 2017 [5]	✗	-	✓	D	Non-RDF	✓	✗
Dia et al. 2018 [17]	Syntactically	✗	✓	D	RDF	✗	✓
Tuli et al. 2019 [42]	✗	-	✓	D	Non-RDF	✓	✗
Cao et al. 2019 [12]	✗	✓	✓	D	Non-RDF	✓	✗
Al-Baltah et al. 2020 [3]	✗	-	-	C	RDF	-	✗
Mebrek et al. 2020 [29]	✗	✓	✓	D	RDF	✓	✗
Ortiz et al. 2022 [32]	✗	-	-	D	Non-RDF	✓	✗
Bonte et al. 2023 [9]	Syntactically	-	✓	D	RDF	✓	✓
DiSIF (our solution)	**Semantically**	✗	✓	**D**	**RDF**	✓	✓

Note: C and D refer to "Centralized" and "Distributed", respectively

tecture—edge, fog, and cloud—specifically designed to support the complex and dynamic data environment of a Digital Twin of a City. As a case study, we apply this model to the traffic detection problem, a critical use case in urban digital twins for real-time monitoring and management.

3.1. Overview of DiSIF Framework

The DiSIF framework consists of three hierarchical layers—edge, fog, and cloud—engineered to facilitate both time-sensitive and complex dependent decision-making processes in IoT applications that underpin Digital Twins of Cities. This layered design enables progressive data processing and fusion aligned with the multi-scale and multi-domain nature of urban digital replicas. We analyze the DiSIF framework's layers from two complementary perspectives: the semantic JDL fusion model and the fusion process itself.

3.1.1. The JDL model perspective

As illustrated in Figure 4, the DiSIF framework is organized into three layers: edge, fog, and cloud. Each layer comprises two types of nodes: worker nodes and master nodes. Worker nodes are responsible for receiving and performing initial processing on data streams collected from physical sensors or lower layers. The processed results are then transmitted to the corresponding master nodes for further fusion and decision-making.

At the edge layer, DiSIF performs the first level of processing, known as object refinement, which involves real-time, resource-efficient operations with minimal latency. This layer is critical in the Digital Twin context for immediate, localized decision-making, such as detecting individual vehicles or traffic incidents. The tasks at this layer align with the object refinement component of the semantic JDL model.

Decisions and fused information from the edge layer are forwarded to the fog layer for the second level of processing, called situation refinement. Here, more complex and aggregated decisions are made, such as identifying traffic congestion patterns or emergent urban events. The fog layer utilizes the cityOnto ontology to semantically integrate and aggregate data, enabling richer context-aware decision-making that reflects the evolving state of the Digital Twin.

Finally, for comprehensive, city-wide analysis and strategic decision-making, the aggregated data and intermediate results from the fog layer are sent to the cloud layer. This layer corresponds to the third level of the JDL model, known as threat refinement or macro-level decision-making. The cloud layer maintains a centralized database storing all processed information and executes intensive computational tasks, including long-term traffic pattern prediction, anomaly detection, and policy-level urban management decisions. Thus, data processing, fusion, and decision-making occur progressively and hierarchically across the DiSIF layers, reflecting the multi-scale nature of the Digital Twin of a City.

3.1.2. The Fusion Model Perspective

This section discusses the types of queries and fusion processes within the DiSIF framework, which is specifically designed to support efficient and scalable data processing across the edge, fog, and cloud layers of a Digital Twin of a City.

Independent queries in C-SPARQL can be executed in parallel without dependencies on other queries. These queries are ideal for tasks such as counting vehicles in different city zones, where sub-queries can run concurrently on multiple distributed nodes, enabling real-time responsiveness essential for Digital Twin applications.

Dependent queries, on the other hand, rely on the results of prior queries and are crucial for complex, sequential operations such as advanced data fusion or layered analysis within the Digital Twin. This capability supports dynamic and adaptive decision-making processes that characterize smart urban environments.

Within the DiSIF framework, data flows hierarchically from the edge layer to the cloud layer, becoming progressively more abstract and semantically enriched. The edge layer handles raw sensor data (level one, object refinement), performing immediate, localized processing. The fog layer processes aggregated information streams (level two, situation refinement), integrating data to identify urban events like traffic congestion. Finally,

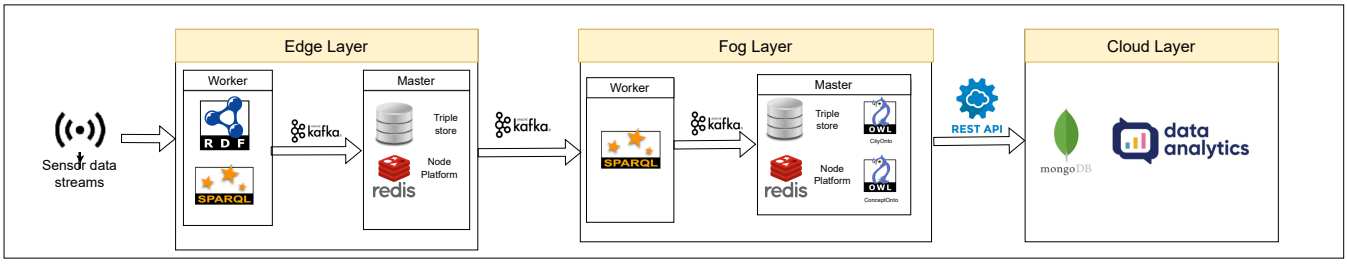

Figure 3: DiSIF Communication

the cloud layer conducts large-scale, strategic decision-making (level three, threat refinement) based on fully integrated city-wide data, supporting the comprehensive management functions of the Digital Twin.

The DiSIF framework, illustrated in Figure 4, can be analyzed from the following perspectives:

Data Level: The framework manages data at increasing levels of abstraction—from raw sensor inputs at the edge to integrated, city-wide information at higher layers. The edge focuses on sensor fusion and immediate event detection, while fog and cloud layers address broader, more complex queries such as urban traffic patterns and predictive analytics.

Query Types: Independent queries are executed in parallel across layers and nodes, enhancing scalability and reducing latency. Dependent queries are executed sequentially within layers, where outputs from one query serve as inputs to subsequent queries, enabling complex workflows necessary for Digital Twin operations.

Fusion Operations: Vertical fusion aggregates homogeneous data across scales (e.g., combining street-level traffic data into area-wide traffic flows), optimizing data volume and query efficiency.

Horizontal fusion integrates heterogeneous data streams (e.g., merging street congestion with weather data) to generate enriched semantic concepts without changing data scale, enabling deeper situational understanding within the Digital Twin.

JDL Model Alignment: DiSIF follows the semantic JDL model, with object refinement at the edge layer, situation refinement at the fog layer, and threat refinement at the cloud layer. This layered, distributed approach supports scalable, real-time decision-making aligned with the multi-level nature of Digital Twins of Cities.

This distributed fusion model enhances data processing efficiency, scalability, and privacy, making it highly suitable for the complex and data-intensive environment of Digital Twins in smart cities. The next section provides a detailed overview of the Edge layer within the DiSIF framework, the foundation for distributed semantic JDL fusion.

3.2. The DiSIF Framework: A Bridge Between the JDL Model and Digital Twin

A key innovation of the DiSIF framework is its structured application of the Joint Directors of Laboratories (JDL) information fusion model within a three-layer distributed architecture

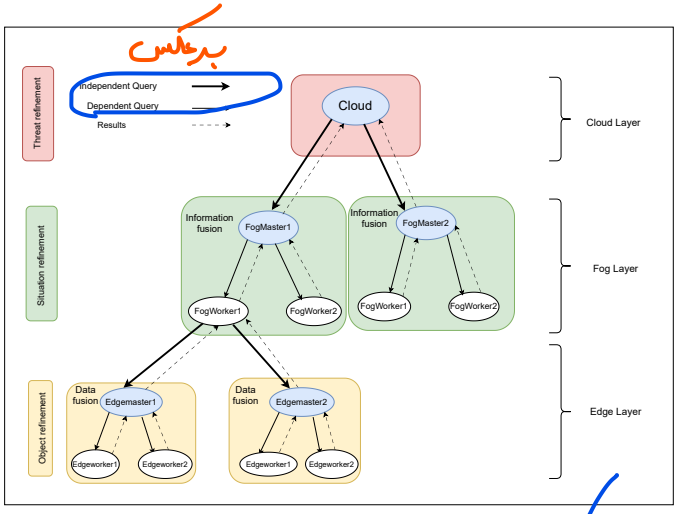

Figure 4: The JDL and fusion model perspectives

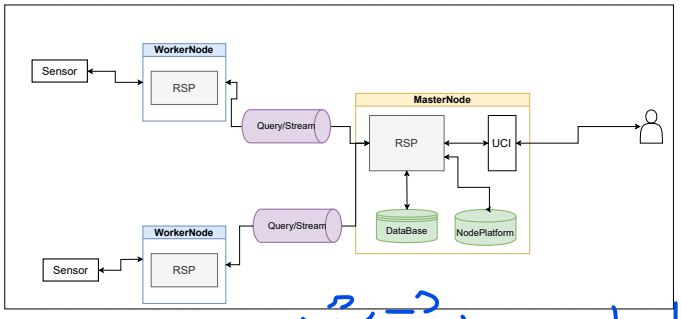

Figure 5: The Edge Layer

(edge, fog, and cloud). This approach underpins the creation and maintenance of a comprehensive, real-time Digital Twin in complex environments such as smart cities. As a dynamic virtual representation of a physical system, a Digital Twin requires a continuous stream of processed data at various levels of abstraction. The DiSIF framework systematically addresses this requirement by aligning its architectural layers with the functional levels of the JDL model.

Level 1: Object Refinement at the Edge

The Edge Layer in the DiSIF architecture is responsible for initial, real-time processing tasks. It directly interfaces with sensors, transforming raw data streams into meaningful initial information. This process directly corresponds to JDL Level 1, Object Refinement. At this level, physical objects (e.g., vehicles,

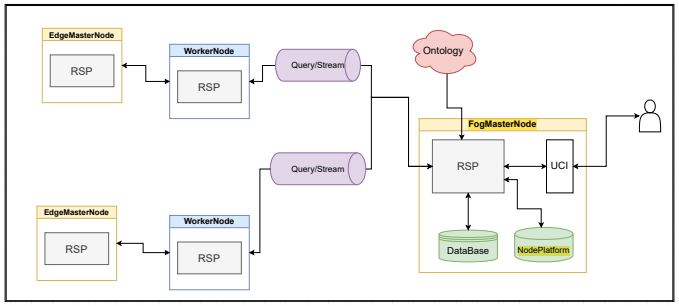

Figure 6: The Fog Layer

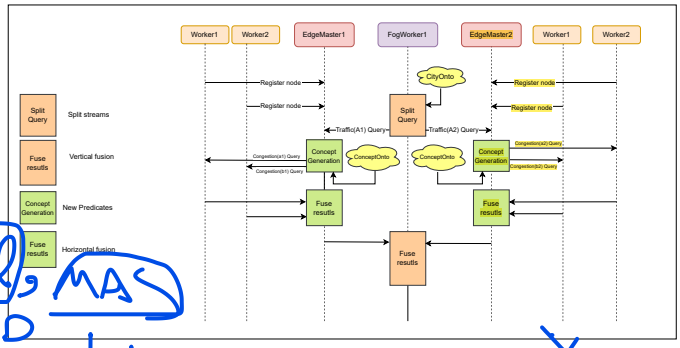

Figure 7: Query execution request flow

environmental sensors) are identified, and their basic attributes (e.g., location, speed, ID) are extracted and converted into a standard RDF format.

Role in the Digital Twin: This layer forms the foundation of the Digital Twin. By processing data closest to the source, an initial, real-time digital representation of each object in the physical environment is created. This constitutes the birth of individual digital entities within the Digital Twin.

Level 2: Situation Refinement in the Fog

The Fog Layer acts as an intermediate tier, receiving processed information from edge nodes and fusing it to achieve a higher-level understanding of the overall situation. This function is equivalent to JDL Level 2, Situation Refinement. In this layer, using ontologies such as CityOnto, the relationships between different objects are analyzed to identify more complex events, such as "heavy traffic" or "congestion".

Role in the Digital Twin: The fog layer enriches the Digital Twin by adding context awareness and an understanding of interactions. At this stage, the Digital Twin evolves from a collection of discrete objects into an integrated, intelligent virtual system capable of reflecting dynamic situations and the complex relationships between a smart city's components.

Level 3: Threat Refinement in the Cloud

The Cloud Layer, as the highest architectural tier, utilizes the enriched data from the fog layer for macro-level analysis, long-term prediction, and strategic decision-making. This level of processing aligns with JDL Level 3, Threat Refinement. By leveraging historical data and live information streams, this layer can forecast complex patterns and assess the potential impact of various scenarios.

Role in the Digital Twin: This layer transforms the Digital Twin into a powerful simulation and forecasting tool. Here, the virtual model not only represents the present state but can also be used to test "what-if" scenarios and optimize city-wide management, moving from a reactive monitor to a proactive optimization engine.

3.3. *Horizontal and Vertical Fusion: Building a Comprehensive and Multi-Resolution Digital Twin*

The two core fusion mechanisms in the DiSIF framework, Vertical Fusion and Horizontal Fusion, play complementary roles in enriching the Digital Twin. These approaches endow the Digital Twin with both analytical depth (through multiple resolutions) and conceptual breadth (by integrating diverse domains), transforming it from a simple representative model into a dynamic and comprehensive analytical tool.

3.3.1. *Vertical Fusion: Enabling a Multi-Resolution Digital Twin*

Vertical fusion, as defined within the DiSIF framework, aggregates homogeneous data at varying scales without changing the core concept of the data. For instance, traffic data from multiple individual streets are combined to generate the traffic flow for a broader area.

This process is vital for creating a **Multi-Resolution Digital Twin**. This capability allows various stakeholders (e.g., city managers, traffic engineers, and citizens) to view the city at different levels of abstraction according to their needs. For example, a traffic control operator can "zoom in" to the granular data of a specific intersection, while an urban planner can "zoom out" to analyze macro-level traffic patterns across a district or the entire city. Therefore, by providing perspectives from the most detailed to the most abstract levels, vertical fusion provides the Digital Twin with analytical depth and flexibility.

3.3.2. *Horizontal Fusion: Generating Urban Insights in a Comprehensive Digital Twin*

Horizontal fusion generates new, richer concepts by integrating heterogeneous data streams from different domains. This process changes the concept of the data; for example, data on "vehicle movement" and "congestion" are combined to create the new concept of "traffic". The DiSIF framework can fuse data from various concepts, such as traffic and weather, to generate new concepts or natures.

This type of fusion is essential for achieving a Comprehensive Digital Twin, as it allows the model to generate novel Urban Insights and to model complex phenomena that arise from the interaction of different systems. For instance, by fusing traffic data with air pollution data, the Digital Twin can model and analyze the direct relationship between traffic jam formations and rising pollutant levels in a specific zone. This capability transforms the Digital Twin from a collection of siloed data representations into a holistic and integrated model of the urban ecosystem, which is critical for evidence-based decision-making.

Table 2:
Example of NodePlatform

Node	Concept	Location	Master
N_1^w	*Congestion*	*loc*1	N_1^m
N_1^w	*Congestion*	*loc*2	N_1^m
N_2^w	*Traffic*	*loc*3	N_3^m
N_3^w	*Congestion*	*loc*5	N_2^m

3.4. Edge Layer

An overall view of the DiSIF framework's Edge layer is depicted in Figure 5.

At the Edge layer, data processing and fusion operations are performed directly on sensor data streams, providing the first line of real-time analysis in the Digital Twin of the City. Worker nodes at this layer are responsible for receiving raw data from sensors, executing assigned queries from the master node, and transmitting processed result streams back to the master node. Master nodes at the edge layer manage user queries received via the User Communication Interface (UCI), store data in the local database, and maintain a registry of worker nodes through the NodePlatform. They orchestrate query execution by assigning sub-queries to worker nodes, aggregating results, and ensuring efficient data flow upwards to the fog layer.

NodePlatform

The NodePlatform, as detailed in Table 2, facilitates the registration and management of worker nodes and their corresponding master nodes within the Edge layer. It stores essential metadata such as node identifiers, generated semantic concepts, and the locations or data streams each node handles. This platform is crucial for decomposing complex queries into sub-queries, dispatching them to appropriate worker nodes, and collecting the resulting data streams at the master node. It ensures that queries are accurately routed to nodes responsible for generating specific semantic concepts within designated urban locations, a key requirement for the spatially distributed nature of Digital Twins. The database component within the master node stores incoming data streams from worker nodes and supports query execution. The UCI acts as the gateway for user interactions, receiving queries and returning results, thereby centralizing user communication.

Algorithm 1 RegisterWorkerNode

1: **procedure** RegisterWorkerNode(*workerNode*)
2: *WorkerNode* publish *joinRequest* message with metadata to Kafka
3: Receive *joinRequest* message at *masterNode*
4: *masterNode* store *workerNode*'s metadata in *Nodeplatform*
5: *masterNode* send *joinResponse* message to *workerNode* with *masterNode* information
6: Receive *joinResponse* at *workerNode* and store *masterNode*'s information
7: **end procedure**

Metadata Information Exchange

The metadata information transmitted from each worker node to its corresponding master node includes: the unique identifier of the *workerNode*, the set of semantic concepts that the *workerNode* can generate, and the specific locations or data streams it is capable of producing. This metadata exchange is fundamental in the context of a Digital Twin of a City, where spatially distributed nodes represent different urban areas or sensor clusters, enabling precise and context-aware data fusion and query routing.

At the Edge layer, horizontal fusion operations—aligned with the object refinement phase of the semantic JDL model—are performed on incoming data. Specifically, the Edge layer collects and preprocesses raw data from physical city components, such as traffic lights, smart waste bins, or weather sensors embedded in specific buildings. This preprocessing, conducted at the object refinement level, transforms raw sensor data into structured semantic representations (e.g., RDF triples) to enable real-time updates of individual object states within the Digital Twin, such as the status of a specific vehicle or the temperature at a particular urban location. These fused and processed streams are then forwarded to worker nodes in the Fog layer for higher-level fusion and decision-making. Algorithm 1 outlines the procedure for assigning worker nodes to their respective master nodes, ensuring efficient task distribution and data management within the Edge layer.

3.5. Fog layer

An overall view of the DiSIF framework's Fog layer is illustrated in Figure 6. Within the Fog layer, data is processed and fused at the level of concept streams rather than raw data streams, reflecting a higher level of semantic abstraction essential for the Digital Twin of a City. Similar to the Edge layer, worker nodes in the Fog layer receive these concept streams from the Edge layer and execute queries assigned by their master nodes.

Master nodes at the Fog layer handle multiple responsibilities: receiving user queries via the User Communication Interface (UCI), storing and managing data in the local database, registering and retrieving information about worker nodes through the NodePlatform, assigning queries to worker nodes, receiving processed data streams, and performing fusion operations on the aggregated streams.

The component architecture of the Fog layer closely mirrors that of the Edge layer but operates on semantically richer data. For vertical fusion, particularly in intelligent traffic management scenarios within the Digital Twin, the cityOnto ontology plays a crucial role as an external semantic reference. This ontology enables the Fog layer to integrate and interpret diverse urban data streams, supporting situation refinement and contextual decision-making. Within the master node, horizontal fusion completes the situation refinement phase by combining heterogeneous concept streams to generate comprehensive, higher-level urban insights.

Query Processing in the Fog Layer

Algorithm 4 details the query response process within the master node of the Fog layer. The master node handles two primary query types: **SPARQL queries** (line 4), which address static queries based on stored database information, and

C-SPARQL queries (line 6), which manage continuous queries over streaming data, essential for real-time Digital Twin operations. At line 7, the algorithm checks if the semantic concepts referenced in the query have been registered by any node within the NodePlatform. If such a node exists, line 8 compares the node's registered locations/streams with those specified in the query. When the requested locations are registered, the query is decomposed into sub-queries based on location (if necessary), and these sub-queries are dispatched to the appropriate nodes. The master node then aggregates the results from these distributed executions (lines 9 to 11). Algorithm 2 further elaborates on query execution: if the node corresponding to the concept and location extracted from the user's query is found, data retrieval, query execution, and result return are performed (lines 6 to 8). If no matching node is found (line 10), the system selects an alternative node from those previously registered in the NodePlatform for the relevant master node and requested location/stream. Subsequently, lines 11 and 12 utilize the ConceptOnto ontology to extract all necessary elements to construct a query capable of generating the new semantic concept. Line 13 executes query construction via Algorithm 3. Finally, line 14 updates the NodePlatform to include the new concept for the selected node and location, and line 15 dispatches the constructed query to the selected node for execution.

The Fog layer aggregates and integrates processed data from the Edge layer to perform situation refinement, a critical step in generating higher-level insights for the Digital Twin. This involves combining concept streams to identify complex urban patterns, such as traffic flow trends across a district, energy consumption profiles of a building block, or air quality variations in a neighborhood. By leveraging the cityOnto ontology for vertical fusion and horizontal fusion for situation refinement, the Fog layer produces semantically enriched insights that enhance contextual decision-making. These fused streams are then forwarded to the Cloud layer for city-wide analysis and strategic decision-making.

3.6. *Cloud layer*

In the Cloud layer of the DiSIF framework, semantic fusion operations are performed on the aggregated concept streams received from the Fog layer, enabling threat refinement at a macro, city-wide level within the Digital Twin of a City. This layer acts as the central hub where global concept streams are stored in a comprehensive database, supporting long-term analysis and predictive modeling essential for strategic urban management. Heavy computational tasks, such as traffic prediction and anomaly detection, are executed periodically or on-demand using both historical data and continuous streams from the Fog layer. Traffic prediction, for example, forecasts congestion and flow patterns across various city locations, providing critical insights for proactive traffic management and urban planning within the Digital Twin environment.

The Cloud layer further supports complex simulations and threat refinement processes, enabling long-term forecasting of city-wide traffic patterns, simulation of urban development scenarios, or analysis of the impact of specific events on urban infrastructure. By leveraging these capabilities, the Cloud layer facilitates data-driven decision-making for strategic urban management, ensuring the Digital Twin remains a robust tool for city-wide planning and resilience.

3.7. *Query Management: Ensuring the Real-Time Viability of the Digital Twin*

One of the most critical features of a Digital Twin is its ability to remain synchronized in real-time with its physical counterpart. This synchronization requires the rapid and efficient processing of massive data streams. The DiSIF framework directly addresses this challenge by providing an optimized query execution model that distinguishes between Independent Queries and Dependent Queries, thereby helping to maintain the real-time nature of the Digital Twin.

3.7.1. *Parallel Execution of Independent Queries for Rapid Updates*

Many update operations within a Digital Twin, such as fetching the instantaneous status of thousands of distributed sensors or assets across a city, are inherently independent of one another. The DiSIF framework has the capability to break down an independent query into multiple sub-queries and execute each one in a parallel and efficient manner across different nodes (Worker Nodes). Experimental results demonstrate that this distributed approach significantly optimizes query execution time.

This parallel execution translates to a substantial reduction in the time required to aggregate data and refresh the overall state of the Digital Twin. Consequently, the latency between an event occurring in the physical world and its reflection in the virtual model is minimized, which underpins the "real-time" nature of the Digital Twin.

3.7.2. *Efficient Handling of Dependent Queries for Complex Simulations*

Complex simulations and deep analytics in a Digital Twin often require a chain of operations, where the output of one query serves as the input for the next. These dependent queries are essential for modeling multifaceted urban phenomena (e.g., analyzing the impact of traffic congestion on air pollution).

In a centralized approach, this chain would be executed sequentially on a single node, leading to a linear increase in response time. However, the DiSIF architecture optimizes this process by executing the prerequisite queries on worker nodes and sending only the processed (and smaller volume) results to the master node for the final dependent query execution. This distributed model eliminates the sequential processing bottleneck and significantly reduces the total execution time for the entire chain. This capability allows the Digital Twin to perform complex, multi-step simulations without falling out of sync with the real world, providing timely and profound insights for decision-makers.

3.7.3. *Support for Dynamic and Complex Queries for an Adaptive Digital Twin*

For a Digital Twin to be an effective management and analytical tool, it cannot be limited to executing only predefined

queries. The urban environment is dynamic and unpredictable, constantly presenting new scenarios and unforeseen analytical needs. The DiSIF framework addresses this critical requirement by supporting

complex and dependent queries that are not predefined and can be introduced at runtime.

Unlike many systems that focus on executing predefined tasks or queries, a complex query can be introduced into the DiSIF architecture at any moment by a user or another service. These types of queries require the results of other queries that have been previously executed in the system to run. The DiSIF architecture manages this process optimally.

Dynamic Execution within Layers: The execution of these new, complex queries is handled by the Master Nodes in each layer, while their prerequisites (pre-queries) are executed by the Worker Nodes of the same layer.

Automatic Query Construction: If a new query requires a concept that is not yet defined in the system, the DiSIF framework leverages a Concept Ontology (ConceptOnto) to automatically extract the necessary elements and construct a new query to generate that concept.

This capability provides the Digital Twin with a high degree of adaptability. For instance, when faced with an unforeseen crisis (such as an extreme weather event or a sudden public health issue), city managers can define and execute entirely new analytical queries to assess the cross-system impacts (e.g., the effect of rainfall on traffic and access to emergency services). DiSIF's ability to dynamically process such queries ensures that the Digital Twin remains a living, evolving tool capable of responding to emergent urban challenges.

3.8. *Query Execution Request Flow*

Figure 7 illustrates a typical query execution scenario involving one FogWorker node and two EdgeMaster nodes, labeled $FogWorker1$, $EdgeMaster1$, and $EdgeMaster2$. Initially, Edge worker nodes register and store their metadata—including generated streams and semantic concepts—in the NodePlatform of their respective EdgeMaster nodes. This registration enables precise query routing and efficient data management aligned with the spatial and conceptual distribution of the Digital Twin.

A user query submitted through FogWorker1 references two independent sub-queries, $Traffic(A1)$ and $Traffic(A2)$, derived from the CityOnto ontology. These sub-queries are executed in parallel on two distinct locations or streams,$A1$ and $A2$, using $EdgeMaster1$ and $EdgeMaster2$, respectively.

Within the EdgeMaster nodes, the concept generation component (Algorithm 3) leverages the ConceptOnto ontology (Figure 8) to generate new semantic concepts that do not have direct incoming data streams. This process results in dependent sub-queries such as $Congestion(a1)$ and $Congestion(b1)$, which are executed on Edge worker nodes. The results are then returned to the corresponding EdgeMaster node.

Horizontal fusion is performed within the Fuse results component of the EdgeMaster nodes, integrating the outputs of these sub-queries. The fused results are subsequently transmitted back to $FogWorker1$, where vertical fusion integrates the outputs

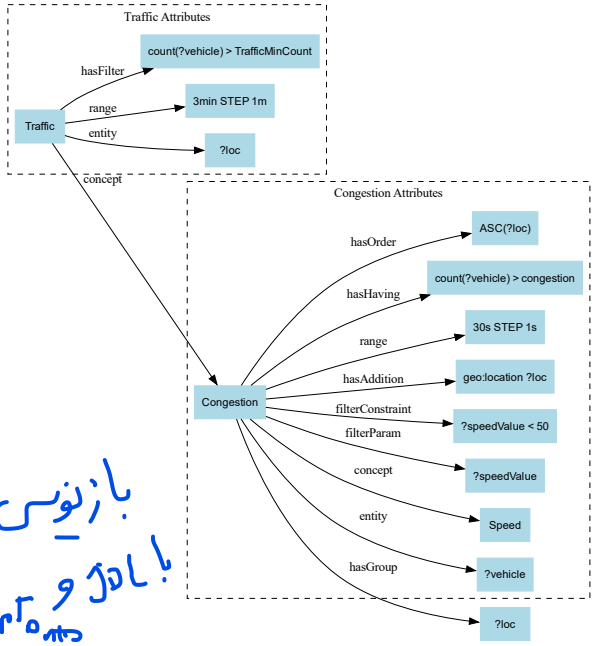

Figure 8: Concept Ontology

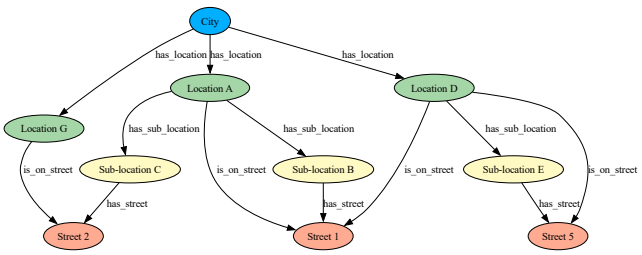

Figure 9: CityOnto example

from streams $A1$ and $A2$. This hierarchical fusion process culminates in generating the final, comprehensive output that reflects the real-time state of the Digital Twin of the City.

4. Evaluation

The DiSIF framework is implemented in the Java programming environment. The codes written for the semantic nodes are independent of the communication layer, allowing the use of various communication channels such as MQTT or WebSocket. The communication channel in the DiSIF framework is based on Apache Kafka. In the system implementation, we utilize C-SPARQL as the RDF stream processor (RSP). Next, we explore the evaluation of the centralized JDL approach and the distributed JDL approach.

As mentioned, in the centralized JDL fusion model, generating the desired outputs requires collecting all the necessary data in a central node, where queries and corresponding components are executed on these aggregated data to produce the outputs.

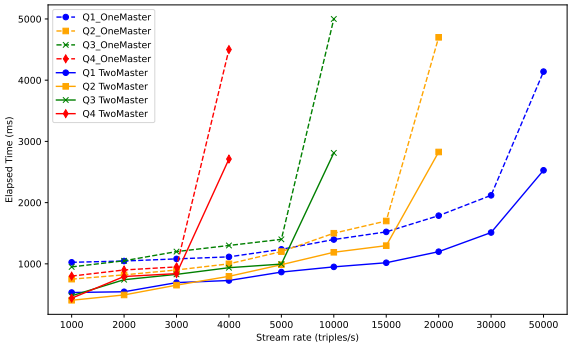

Figure 10: Object refinement performance: independent query execution time for different master nodes

In contrast, in the distributed JDL approach, there is no need to send all data to a central node. Instead, by distributing query processing, only the query execution results are sent to other nodes.

Comparison of centralized and distributed JDL approaches can be analyzed from five perspectives:

- **Network Load**

  In the centralized approach, since all RDF data needs to be sent to the master node, the network load increases significantly, leading to a decrease in network efficiency. The volume of raw data sent over the network to the master node is much larger than the processed data. In terms of the amount of data transmitted and data transfer speed, the approach of sending processed data is preferable to sending raw data. This makes the distributed JDL approach more network-efficient than the centralized JDL approach.

- **Execution Time of Dependent Queries**

  In the JDL model, the situation refinement component, requiring the use of outputs from the object refinement component, executes dependent queries for output generation. As subqueries need to be executed first to provide the necessary input for dependent queries, the time to produce outputs for dependent queries increases. In the centralized approach, both subqueries and dependent queries are executed on a single node, while in the distributed JDL approach, subqueries are executed on worker nodes, and dependent queries are executed on the master node. Consequently, the execution time of dependent queries is reduced, making it more efficient compared to the centralized approach.

- **Execution Time of Independent Queries**

  In the JDL model, the object refinement component includes independent queries that operate on raw data and do not require the execution of other queries as prerequisites. In the distributed approach, these queries can be executed across different nodes, enhancing the execution time of independent queries compared to the centralized scenario.

Algorithm 2 Process Query

```
1:  procedure ProcessQuery(query)
2:      while True do
3:          Concepts, Locations ← ExtractConceptsLocations(query)
4:          node ← findNode(Concepts, Locations)
5:          if node exists then
6:              ListenOnNode(node)
7:              result ← Execute(query)
8:              return result
9:          else
10:             node ← SelectWorkerNode()
11:             Entity, Range, hasAddition, ConceptNew, filterParam, filterConstraints, hasGroup, hasHaving, hasOrder ← GetParameterFromConceptOnto(Concepts)
12:             NewQuery ← ConstructQuery(Concepts, Locations, Entity, Range, hasAddition, ConceptNew, filterParam, filterConstraints, hasGroup, hasHaving, hasOrder)
13:             UpdateNodePlatform(Concepts, Locations, node)
14:             SendQueryToNode(NewQuery, node)
15:         end if
16:     end while
17: end procedure
```

- **Memory Consumption**

  In the centralized JDL model, the execution of independent and dependent queries on a single node can significantly impact their memory consumption. On the other hand, the distributed JDL approach has demonstrated better memory management compared to the centralized approach.

- **Data Security**

  One of the challenges of the centralized JDL method is that all data must be sent from other nodes to the master node, posing potential security issues. In many applications, data from nodes cannot be transferred to the master node due to security concerns and must be used locally. Therefore, the distributed JDL approach is introduced to overcome this challenge. In this approach, there is no need to send raw data from other nodes to the central node, and data processing can be performed locally on local data, with the results sent to the master node. Thus, the security issue related to data transfer is mitigated in this distributed approach.

In this section, we analyze the centralized and distributed JDL approaches in terms of executing various JDL components. To evaluate the object refinement component in the edge layer and the situation refinement component in the fog layer, we analyze the executing of independent and dependent queries, respectively, in both centralized and distributed scenarios.

Algorithm 3 Construct Query

1: **procedure** ConstructQuery(*Concept*, *Location*, *Entity*, *Range*, *hasAddition*, *ConceptNew*, *filterParam*, *filterConstraints*, *hasGroup*, *hasHaving*, *hasOrder*)
2: *query* ←
```
CONSTRUCT {?l concept:{Concept} ?a}
FROM STREAM {Location} [RANGE {Range}]
     WHERE { {Entity} concept:{ConceptNew}
{filterParam},
{Entity} {hasAddition},
    FILTER ({filterConstraints})
}
GROUP BY {hasGroup}
HAVING {hasHaving}
ORDER BY {hasOrder}
```
3: **end procedure**

Algorithm 4 Query Response in masterNode

1: **Input:** User query
2: **Output:** Query response
3: Receive user query by UCI and send to RSP component
4: **if** SPARQL query **then**
5: Execute query on database and get results.
6: **else if** C-SPARQL query **then**
7: **if** Query's concepts exist in NodePlatform **then**
8: **if** Query's locations exist in NodePlatform **then**
9: Split the query by locations into independent sub-queries.
10: Assign each sub-query to corresponding node registered in NodePlatform.
11: Aggregate results of sub-queries.
12: **else**
13: Expand each query stream/location with its sub-streams/sub-locations according to the cityOnto.
14: Repeat the steps from line 6.
15: **end if**
16: **else**
17: Create a new query for generating the desired concept based on Algorithm 2 using the conceptOntology.
18: **end if**
19: **end if**

4.1. Object refinement performance in the Edge layer

The data in this layer consists of sensor data (level one), and the queries processed at this level are classified as level one or independent queries. Consequently, the fusion operation occurs at the sensor level, referred to as sensor/data fusion. Subsequently, the performance of the DiSIF framework is analyzed in terms of the execution time of level one/independent queries in the edge layer.

4.1.1. Centralized and Distributed approaches

In these experiments, we analyze the time required to execute queries Q1, Q2, Q3, and Q4 (shown in Appendix A) from the perspective of the stream rate.

in Figure 10 , the results of executing various queries in the centralized scenario, where only one edge master node exists and for distributed approach with two edge master nodes, are presented. In this case, the execution time of queries is analyzed for different stream rates. As observed, for query Q1, the execution time experiences a sudden increase at a stream rate of 30000 triples/s. Similarly, query Q2 shows a sudden increase at a stream rate of 15000 triples/s, Q3 at 5000 triples/s, and finally, Q4 at 3000 triples/s.

The reason for this phenomenon is that, in C-SPARQL, as the complexity of a query increases, its ability to manage high stream rates decreases. When the stream rate exceeds the response capacity of C-SPARQL, the execution time of the query experiences a sudden increase. For this reason, in cases where a query is broken down into independent subqueries and multiple edge master nodes are available for query execution, each subquery can be executed on a separate edge master node and the results are then combined, enabling the main query to be executed in parallel. Consequently, the execution time of queries significantly improves in distributed approach.

As depicted in Figure 10, the execution time for different stream rates is approximately halved. The reason for this slight increase in the execution time is that a short period is spent aggregating the results from these two nodes. Therefore, in comparison to the centralized approach for executing independent queries, this distributed scenario exhibits lower execution times.

4.2. Situation refinement performance in the Fog layer

In this section, we evaluate the performance of the distributed JDL fusion model in comparison to the centralized JDL, focusing specifically on the situation refinement component and the execution of dependent queries. The scenarios of congestion detection and traffic discovery are examined as two dependent scenarios or queries.

The data flowing between nodes in the fog layer is categorized as level two. In other words, this data consists of processed information from the lower edge layer, rather than raw sensor data. Consequently, the fusion operation at this level involves information fusion, and the queries executed by fog layer managers are focused on concepts that require prepared input data for their execution.

In the centralized JDL approach, obtaining the results of the situation refinement component requires the outputs of the object refinement component to be initially placed on the BUS associated with the centralized JDL. Consequently, object refinement and situation refinement queries are interdependent and must be executed in a sequential manner.

In the following, we evaluate the two components, object refinement and situation refinement, for the scenarios of congestion detection and traffic discovery, respectively.

Traffic Discovery Scenario

For traffic discovery, the query Q_m is defined in Appendix A .

As indicated by query Q_m, congestion event messages are received within 3-second windows. In this query, `?s` represents the streets (as locations) where congestion has occurred. If a street experiences congestion more than three times within a 3-second window, it is classified as congested. To execute this query, congestion event messages must be generated, which are produced by worker nodes within the same fog layer.

Congestion Detection Scenario

To evaluate the performance of both centralized and distributed JDL approaches for congestion detection, we employ various queries with diverse complexities as detailed in the Appendix A as Queries Q_1,Q_2,Q_3,Q_4.

Query Q_1

In this query, the output highlights regions where the speed of at least one vehicle is below 50 km/h, indicating congestion. This query is specifically designed to detect congestion and generate congestion event messages based on the location, speed, and timestamp of vehicles within the specified stream.

Query Q_2

Continuing with the congestion detection scenario, Query Q_2 identifies regions where at least three vehicles have speeds below 50 km/h, indicating congestion. The results are then sorted in ascending order based on location. This query establishes a more specific criterion for detecting congestion by taking into account both the speed condition and the minimum number of vehicles present in a given area.

Query Q_3

This query, similar to Query Q_2, congestion is detected in areas with "2," "3," or "1" in their titles (?location). Additionally, the average speed of vehicles is returned as output for each location. This query offers insights into both congestion detection and the average speed of vehicles in specific locations.

Query Q_4

This operates similarly to Query Q_3, with the difference that it uses UNION to also analyze areas whose title includes "4". This allows the query to provide insights into congestion detection and average speed in regions containing "2", "3", "1", or "4".

We evaluate the situation refinement component from three perspectives: query execution time, memory consumption, and network load. Each of these aspects will be examined in detail. Assume that executing the user query Q_m requires the execution of n prerequisite queries Q_i.

4.2.1. Query execution time

In this section, we first express the formula for calculating the execution time of query Q_m in centralized and distributed approaches as follows.

- Centralized approach

  In the centralized approach, all raw data must be transferred from the worker nodes to the master node before executing query Q_m on the master node.

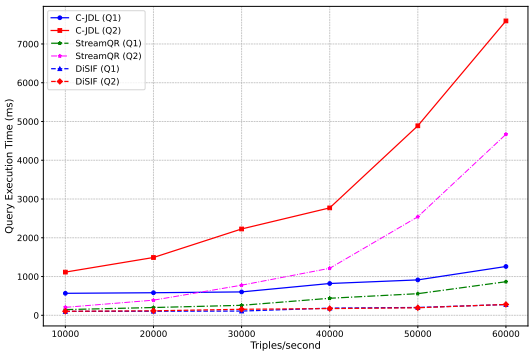

Figure 11: Comparing Execution Times for Q1 and Q2 in Centralized and Distributed Environments (Linear Scale)

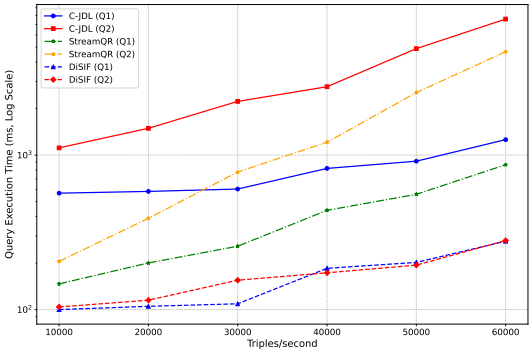

Figure 12: Comparing Logarithmic Execution Times for Q1 and Q2 in Centralized and Distributed Environments

$$T_c = T_{\text{data transfer}} + \sum_{i \in N} T_{Q_i} + T_{Q_m} \tag{1}$$

where $T_{\text{data transfer}}$ is the time required to transfer raw data from all worker nodes to the master node, T_{Q_i} is the time taken to execute query Q_i on the master node, and T_{Q_m} is the time required to execute query Q_m on the master node.

- Distributed approach

  In the distributed approach, each query is executed on its respective worker node, and the results are transferred to the master node, where the query Q_m is executed. The execution time in the distributed approach is as follows:

$$T_d = \max(T_{Q_i}) + T_{\text{result transfer}} + T_{Q_m} \tag{2}$$

As can be observed from the equations, in the centralized approach, the execution time increases linearly with the increase in the number of queries Q_i. This is due to the sequential processing. In the distributed approach, the queries are processed in parallel, which reduces the execution time to the maximum time required to execute any of the Q_i queries. Therefore, for a large number of requests N, the distributed approach significantly reduces the execution time compared to the centralized approach due to parallel execution.

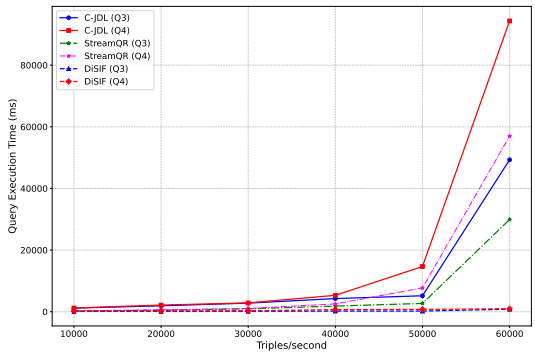

Figure 13: Comparing Execution Times for Q3 and Q4 in Centralized and Distributed Environments (Linear Scale)

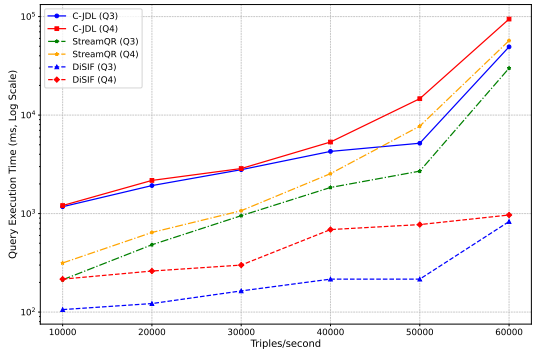

Figure 14: Comparing Logarithmic Execution Times for Q3 and Q4 in Centralized and Distributed Environments

To analyze the execution time for queries Q_1 to Q_4 and subsequently query Q_m, Figures 11,12, 13 and 14 illustrates the results for various data transmission rates (triples per second). This analysis compares both the centralized JDL (C-JDL), StreamQR and DiSIF approaches.

In the C-JDL model, all queries, including Q_1, Q_2, Q_3, or Q_4, and Q_m, are executed sequentially on the master node. For example, during the time analysis of queries Q_1 to Q_4 and Q_m, all raw data is sent from the worker nodes to the master node, where the queries are processed. This centralized structure leads to exponential growth in query execution time, especially for more complex queries (like Q_3 and Q_4), as the data transmission rate (triples/sec) increases. This increase is due to Q_m's dependency on the output of Q_3 and Q_4, both of which must be processed on the master node. As a result, the C-JDL method exhibits the longest query execution time and has relatively poor performance in terms of latency and efficiency.

In the StreamQR model, queries are aggregated and executed as a single large query, which significantly improves execution time compared to the C-JDL method. Since all queries are aggregated and executed in one process, the sequential execution is eliminated, and processing speed increases. However, as the data transmission rate increases, particularly at higher rates, the complexity and size of the aggregated query grow, and its execution time gradually increases. At high rates, the execution time of StreamQR may approach that of C-JDL, especially when the aggregated query becomes very complex and large.

In the DiSIF model, queries are executed locally the worker nodes within the fog layer, and only the processed results are sent to the master node. This significantly reduces query execution time, as parallel processing occurs on the worker nodes, with only the final aggregation (via the execution of query Q_m) performed on the master node. Unlike the previous methods, DiSIF has the shortest query execution time, as it leverages distributed processing across the network rather than relying on a single central node, resulting in much better efficiency.

As observed in Figure 11 and 12, the execution time of DiSIF(Q1), StreamQR(Q1) and C-JDL(Q1) increases almost linearly with the increase in the data sending rate. Moreover, the time needed for executing DiSIF(Q1) is generally less compared to others, and this time difference remains approximately constant across various data sending rates.

Figures 13 and 14 reveal an exponential increase in execution time for queries C-JDL(Q3), StreamQR(Q3), C-JDL(Q4) and StreamQR(Q4) as the data sending rate rises. In contrast, the execution time for DiSIF(Q3) and DiSIF(Q4) shows a much less significant growth rate. This discrepancy is due to the dependency of query Q_m on Q_3 and Q_4. When both queries are executed on the master node (centralized mode), the delay becomes substantially higher. In the DiSIF model, however, queries Q_3 and Q_4 are processed on the worker nodes , and only the results are sent to the master node, thereby reducing delays associated with producing results for Q_m. This illustrates the stability or robustness of the DiSIF method.

4.3. Network load perspective

Next, we compare C-JDL, StreamQR and DiSIF approaches in terms of the number and volume of messages transmitted across the network (network load).

In a centralized approach (C-JDL and StreamQR), the total load L_c is given by the sum of all raw data D_i sent from each worker node i to the master node:

$$L_c = \sum_{i \in N} D_i \tag{3}$$

Here, D_i represents the raw data transmitted from worker node i to the master node.

In DiSIF approach, the total load L_d is given by the sum of all results R_i obtained by each worker node i and sent to the master node:

$$L_d = \sum_{i \in N} R_i \tag{4}$$

In this case, R_i represents the results obtained by worker node i that are transmitted to the master node.

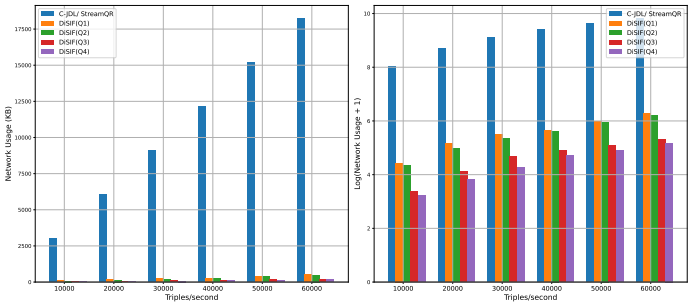

Figure 15: Network usage comparison for different stream rates

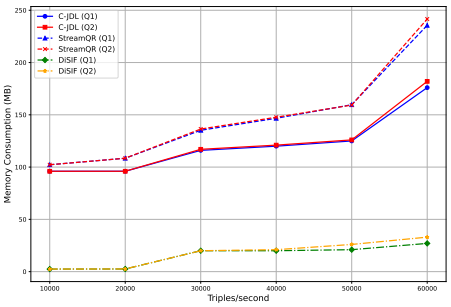

Figure 16: Memory consumption for a single stream receiver node (Linear Scale) for Q1 and Q2

As can be seen from these expressions, for large data transmission, L_c is significantly greater than L_d. Therefore, in the centralized approaches, the network load is high due to the transfer of all raw data from the worker nodes to the master node, whereas in DiSIF approach, the network load is minimized by transferring only the processed results.

An example of a raw RDF message used in the Kafka system for sending from worker nodes To the master node is as follows:

```
https://www.wtlab.com/TrafficStream/vehicle37450
  http://www.w3.org/2003/01/geo/wgs84_pos
   #location
  7103


https://www.wtlab.com/TrafficStream/vehicle37450
 http://example.org/timestamp
 2023-09-20T12:00:028948


https://www.wtlab.com/TrafficStream/vehicle37450
 http://example.org/speed
 75^^http://www.w3.org/2001/XMLSchema#int
```

Each raw message consists of 317 characters, and its size is 311 bytes. Additionally, an example of the output message obtained from queries Q_1 to Q_4, which is sent from worker nodes to the master node in DiSIF approach, is as follows:

```
"7103 congestion"
```

This message contains 15 characters and 14 bytes. In Figure 15, the network usage for sending messages for various queries in C-JDL, StreamQR and DiSIF approaches is illustrated.

As observed in Figure 15 , the volume of messages sent over the network wih Kafka in the C-JDL/StreamQR approaches is significantly higher compared to DiSIF approach.

In C-JDL/StreamQR approaches, all raw RDF data must be transmitted from worker nodes to the master node, resulting in a substantial network load. In contrast, the DiSIF method involves performing local computations and query executions on the worker nodes, with only the processed results—comprising much smaller message volumes—being transmitted to the master node.

Furthermore, as the data streaming rate increases, the number of messages sent also rises, highlighting the distinction between C-JDL/StreamQR approaches and DiSIF. Additionally, as the complexity of the queries increases ($Q_1 < Q_2 < Q_3 < Q_4$), fewer messages are sent in the network. In contrast, with C-JDL/StreamQR approaches, there is no significant difference in the volume of sent messages with respect to the complexity of the queries.

4.4. Memory Consumption Perspective

In terms of memory consumption, we employ the following two formulas to calculate the memory requirements for executing query Q_m in both centralized and distributed approaches:

$$\text{Mem}_c(Q_i) = M_{Q_iQ_m} > M_{Q_i} + M_{Q_m} \tag{5}$$

$$\text{Mem}_d(Q_i) = \max(M_{Q_i}, M_{Q_m}) \tag{6}$$

As observed, $M_{Q_iQ_m}$ signifies the amount of memory required when executing Q_m and Q_i sequentially on a master node. In both approaches, memory consumption for data transmission is neglected.

In the centralized approach, the sequential execution of queries Q_i and Q_m results in the generation of intermediate data in the memory of the master node. This leads to higher memory consumption ($\text{Mem}_c(Q_iQ_m)$) compared to the sum of individual memory consumptions ($M_{Q_i} + M_{Q_m}$).

However, in the distributed approach, since queries Q_i and Q_m are executed in parallel on different nodes, the memory consumption is equal to the maximum of the memory requirements for Q_i and Q_m in both worker and master nodes.

Next, we will analyze the distributed and centralized JDL methods in terms of memory consumption for executing query Q_m on the master node.

The StreamQR method, due to the aggregation of queries and the execution of a single large query (expanded query), can have higher memory consumption compared to the C-JDL method. The execution of this large query may require significant memory to process all the input data simultaneously and store intermediate results. Managing and processing the aggregated query, along with handling large volumes of intermediate data and results, can substantially increase memory usage in StreamQR.

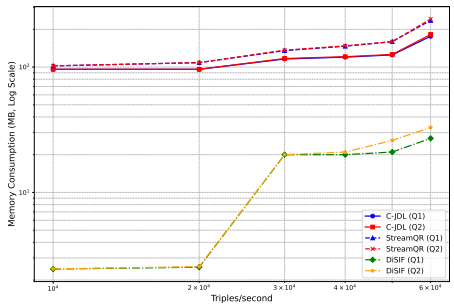

Figure 17: Memory consumption for a single stream receiver node (Log Scale) for Q1 and Q2

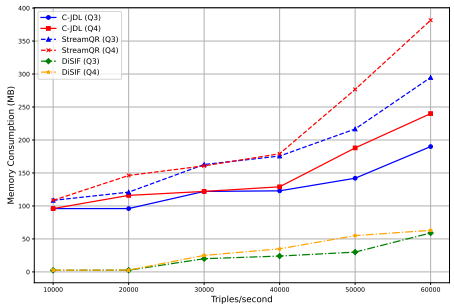

Figure 18: Memory consumption for a single stream receiver node (Linear Scale) for Q3 and Q4

In contrast, the C-JDL method manages memory separately for each query. Memory is temporarily released after each query's execution, as memory is only used for the results and processing of individual queries. Despite this, centralized processing in C-JDL can still result in high memory usage when handling more complex queries, but it is generally lower than StreamQR because queries are executed individually rather than aggregated.

As shown in Figures 16 and 17, memory consumption for executing queries Q_1 and Q_2 in the C-JDL is over four times greater than in the DiSIF. This occurs because, in the C-JDL, Q_1 (or Q_2) must be processed to generate outputs stored in the master node's memory, which are then used as input for query Q_m to obtain the final results. As a result, memory usage in the C-JDL is significantly higher compared to the DiSIF.

Query Q_2, due to its use of aggregator functions such as GroupBy and Having, requires more memory consumption compared to Q_1.

On the other hand, Figures 18 and 19, show that memory consumption for queries Q_3 and Q_4 in C-JDL and StreamQR is significantly higher than in DiSIF approach. This increase is due to the use of AVG functions and CONTAINS, which can elevate memory usage. Storing the outputs in memory and then executing Q_m on these outputs substantially increases memory consumption for Q_3 and Q_4 in C-JDL and StreamQR compared to DiSIF approach.

Moreover, Figures 16 and 18 demonstrate that the AVG and

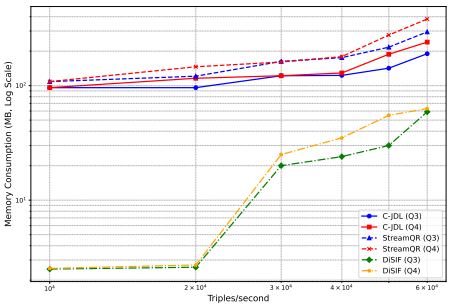

Figure 19: Memory consumption for a single stream receiver node (Log Scale) for Q3 and Q4

CONTAINS functions in queries Q_3 and Q_4 can significantly increase memory consumption, particularly when data transmission rates exceed 40,000 triples per second in C-JDL and StreamQR. Additionally, as the data transmission rate increases, memory consumption for Q_4 surpasses that of Q_3, a trend that becomes noticeable when data transmission rates surpass 40,000 triples per second.

5. Conclusion and future works

This study introduces a novel distributed semantic JDL fusion model tailored for smart city applications, leveraging a three-layer architecture consisting of edge, fog, and cloud layers. Our framework addresses the limitations of centralized fusion models, particularly the inefficiencies associated with processing vast volumes of heterogeneous data in smart cities. Key benefits of our approach include:

Enhanced Network Efficiency: By performing low-level data processing at the edge and transmitting only the processed results to higher layers, our model significantly reduces network load and optimizes bandwidth usage.

Reduced Query Execution Time: The ability to decompose complex queries into independent and dependent sub-queries, executed in parallel across different layers, ensures faster query responses and improves overall system responsiveness.

Improved Data Privacy: Our distributed approach minimizes the need to transmit raw data across the network, thereby enhancing data privacy and security.

Resource Optimization: Distributing computational loads across multiple nodes and layers reduces memory consumption and improves processing efficiency, making the system more scalable and robust.

Our evaluations demonstrate that the distributed JDL model outperforms traditional centralized approaches by reducing network load, decreasing query execution time, and optimizing memory usage. The integration of horizontal and vertical fusion techniques allows for effective management of both heterogeneous and homogeneous data, thus improving the reliability and accuracy of decision-making processes in smart cities.

Furthermore, the DiSIF framework supports real-time, decentralized decision-making, which is critical for addressing the

diverse and dynamic needs of urban environments. The innovative approach of combining data fusion at different layers with a distributed query execution model provides a comprehensive solution for the complex data management challenges faced by smart cities.

Future research will focus on refining the model, exploring its application across various smart city scenarios, and addressing new challenges related to large-scale data fusion and real-time processing. The DiSIF framework represents a significant advancement in the efficient management and utilization of smart city data, paving the way for more responsive, adaptive, and intelligent urban systems.

6. Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] Nasir Abbas, Yan Zhang, Amir Taherkordi, and T Skeie. Mobile Edge Computing: A Survey. *IEEE Internet of Things Journal*, 5:450–465, 2018.

[2] E Ahmed, A Ahmed, Ibrar Yaqoob, Junaid Shuja, A Gani, Muhammad Imran, and Muhammad Shoaib. Bringing Computation Closer toward the User Network: Is Edge Computing the Solution? *IEEE Communications Magazine*, 55:138–144, 2017.

[3] Ibrahim Ahmed Al-Baltah, Abdul Azim Abd Ghani, Ghilan Mohammed Al-Gomaei, Fua'ad Hassan Abdulrazzak, and Abdulmonem Ali Al Kharusi. A scalable semantic data fusion framework for heterogeneous sensors data. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–20, 2020.

[4] Darko Anicic, Paul Fodor, Sebastian Rudolph, and Nenad Stojanovic. Ep-sparql: a unified language for event processing and stream reasoning. In *Proceedings of the 20th international conference on World wide web*, pages 635–644, 2011.

[5] Hamid Reza Arkian, Abolfazl Diyanat, and Atefe Pourkhalili. MIST: Fog-based data analytics scheme with cost-efficient resource provisioning for IoT crowdsensing applications. *Journal of Network and Computer Applications*, 82:152–165, 2017.

[6] E Baccarelli, P Naranjo, M Scarpiniti, Mohammad Shojafar, and J Abawajy. Fog of Everything: Energy-Efficient Networked Computing Architectures, Research Challenges, and a Case Study. *IEEE Access*, 5:9882–9910, 2017.

[7] Davide Francesco Barbieri, Daniele Braga, Stefano Ceri, Emanuele Della Valle, and Michael Grossniklaus. C-sparql: a continuous query language for rdf data streams. *International Journal of Semantic Computing*, 4(01):3–25, 2010.

[8] Matteo Belcao. Streaming virtual knowledge graphs for real-time big data analytics. 2020.

[9] Pieter Bonte and Femke Ongenae. Towards cascading reasoning for generic edge processing. In *ESWC2023, the First International Workshop on Semantic Web on Constrained Things*, 2023.

[10] Jean-Paul Calbimonte, Oscar Corcho, and Alasdair JG Gray. Enabling ontology-based access to streaming data sources. In *The Semantic Web–ISWC 2010: 9th International Semantic Web Conference, ISWC 2010, Shanghai, China, November 7-11, 2010, Revised Selected Papers, Part I 9*, pages 96–111. Springer, 2010.

[11] Jean-Paul Calbimonte, Jose Mora, and Oscar Corcho. Query rewriting in rdf stream processing. In *The Semantic Web. Latest Advances and New Domains: 13th International Conference, ESWC 2016, Heraklion, Crete, Greece, May 29–June 2, 2016, Proceedings 13*, pages 486–502. Springer, 2016.

[12] Hung Cao and M Wachowicz. An Edge-Fog-Cloud Architecture of Streaming Analytics for Internet of Things Applications. *Sensors*, 19:3594, 2019.

[13] Amir Vahid Dastjerdi, Harshit Gupta, Rodrigo N Calheiros, Soumya K Ghosh, and Rajkumar Buyya. Fog computing: Principles, architectures, and applications. In *Internet of things*, pages 61–75. Elsevier, 2016.

[14] Suparna De, Tarek Elsaleh, Payam Barnaghi, and Stefan Meissner. An Internet of Things Platform for Real-World and Digital Objects. *Journal of Scalable Computing: Practice and Experience*, 13:45–57, 2012.

[15] Mathias De Brouwer, Filip De Turck, and Femke Ongenae. Enabling efficient semantic stream processing across the iot network through adaptive distribution with divide. *Journal of Network and Systems Management*, 32(2):27, 2024.

[16] Mario San Emeterio de la Parte, José-Fernán Martínez-Ortega, Pedro Castillejo, and Néstor Lucas-Martínez. Spatio-temporal semantic data management systems for iot in agriculture 5.0: Challenges and future directions. *Internet of Things*, page 101030, 2023.

[17] Amadou Fall Dia, Zakia Kazi-Aoul, Aliou Boly, and Elisabeth Métais. Drss: Distributed rdf sparql streaming. *Software Engineering Research, Management and Applications*, pages 125–145, 2018.

[18] Víctor Delgado García. Exploring the limits of cloud computing. 2011.

[19] Mehdi Gheisari, Hamid Esmaeili Najafabadi, Jafar A. Alzubi, Jiechao Gao, Guojun Wang, Aaqif Afzaal Abbasi, and Aniello Castiglione. Obpp: An ontology-based framework for privacy-preserving in iot-based smart city. *Future Generation Computer Systems*, 123:1–13, 2021.

[20] Syed Gillani, Gauthier Picard, and Frédérique Laforest. Dionysus: Towards query-aware distributed processing of rdf graph streams. In *EDBT/ICDT Workshops*. Citeseer, 2016.

[21] A Giordano, G Spezzano, and A Vinci. Smart Agents and Fog Computing for Smart City Applications. In *Smart-CT*, 2016.

[22] Amelie Gyrard. An Architecture to Aggregate Heterogeneous and Semantic Sensed Data. volume 7882, pages 697–701, 2013.

[23] David L Hall and James Llinas. An introduction to multisensor data fusion. *Proceedings of the IEEE*, 85(1):6–23, 1997.

[24] Pengfei Hu, Sahraoui Dhelim, H Ning, and T Qiu. Survey on fog computing: architecture, key technologies, applications and open issues. *J. Netw. Comput. Appl.*, 98:27–42, 2017.

[25] Houda Khrouf, Badre Belabbess, Laurent Bihanic, Gabriel Képéklian, and Olivier Curé. Waves: big data platform for real-time rdf stream processing. In *SR workshop at ISWC*, 2016.

[26] Srdjan Komazec, Davide Cerri, and Dieter Fensel. Sparkwave: continuous schema-enhanced pattern matching over rdf data streams. In *Proceedings of the 6th ACM International Conference on Distributed Event-Based Systems*, pages 58–68, 2012.

[27] Danh Le-Phuoc, Minh Dao-Tran, Josiane Xavier Parreira, and Manfred Hauswirth. A native and adaptive approach for unified processing of linked streams and linked data. In *International Semantic Web Conference*, pages 370–388. Springer, 2011.

[28] Danh Le-Phuoc, Hoan Nguyen Mau Quoc, Chan Le Van, and Manfred Hauswirth. Elastic and scalable processing of linked stream data in the cloud. In *International Semantic Web Conference*, pages 280–297. Springer, 2013.

[29] Wafaa Mebrek and Amel Bouzeghoub. A stream reasoning framework based on a multi-agents model. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, pages 509–512, 2020.

[30] Ahmad Nagib and Haitham Hamza. SIGHTED: A Framework for Semantic Integration of Heterogeneous Sensor Data on the Internet of Things. *Procedia Computer Science*, 83:529–536, 2016.

[31] Havva Alizadeh Noughabi, Mohsen Kahani, and Behshid Behkamal. Semfus: semantic fusion framework based on jdl. In *Innovations and Advances in Computer, Information, Systems Sciences, and Engineering*, pages 583–594. Springer, 2013.

[32] Guadalupe Ortiz, Meftah Zouai, Okba Kazar, Alfonso Garcia-de Prado, and Juan Boubeta-Puig. Atmosphere: Context and situational-aware collaborative iot architecture for edge-fog-cloud computing. *Computer Standards & Interfaces*, 79:103550, 2022.

[33] Harshal Patni, Cory Henson, Michael Cooney, Amit Sheth, and Krishnaprasad Thirunarayan. Demonstration: Real-Time Semantic Analysis of Sensor Streams. volume 839, 2011.

[34] Charith Perera, Yongrui Qin, J C Estrella, S Reiff-Marganiec, and A Vasilakos. Fog Computing for Sustainable Smart Cities: A Survey. *arXiv: Networking and Internet Architecture*, 2017.

[35] Danh Phuoc, Hoan Nguyen Mau Quoc, Josiane Parreira, and Manfred Hauswirth. A middleware framework for scalable management of linked

streams. *Web Semantics: Science, Services and Agents on the World Wide Web*, 16:42–51, 2012.
[36] M A Rahman, M Rashid, M Hossain, E Hassanain, Mohammed F Alhamid, and M Guizani. Blockchain and IoT-Based Cognitive Edge Framework for Sharing Economy Services in a Smart City. *IEEE Access*, 7:18611–18621, 2019.
[37] Mikko Rinne, Esko Nuutila, and Seppo Törmä. Instans: High-performance event processing with standard rdf and sparql. In *11th International Semantic Web Conference ISWC*, volume 914, pages 101–104. Citeseer, 2012.
[38] Soroush Samadian, Bruce McManus, and Mark D Wilkinson. Automatic detection and resolution of measurement-unit conflicts in aggregated data. *BMC medical genomics*, 7(1):1–8, 2014.
[39] Youssef Sellami, Youcef Imine, and Antoine Gallais. A verifiable data integrity scheme for distributed data sharing in fog computing architecture. *Future Generation Computer Systems*, 150:64–77, 2024.
[40] W Shi, J Cao, Q Zhang, Y Li, and Lanyu Xu. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal*, 3:637–646, 2016.
[41] Eugene Siow, Thanassis Tiropanis, and Wendy Hall. Ewya: An Interoperable Fog Computing Infrastructure with RDF Stream Processing. pages 245–265, 2017.
[42] Shreshth Tuli, Redowan Mahmud, Shikhar Tuli, and Rajkumar Buyya. FogBus: A Blockchain-based Lightweight Framework for Edge and Fog Computing. *Journal of Systems and Software*, 154:22–36, 2019.
[43] Feng Wang, Liang Hu, Jin Zhou, Jiejun Hu, and Kuo Zhao. A semantics-based approach to multi-source heterogeneous information fusion in the internet of things. *Soft Computing*, 21(8):2005–2013, 2017.
[44] Fatos Xhafa, Burak Kilic, and Paul Krause. Evaluation of iot stream processing at edge computing layer for semantic data enrichment. *Future Generation Computer Systems*, 105:730–736, 2020.
[45] Anastasios Zafeiropoulos, Nikolaos Konstantinou, Stamatios Arkoulis, Dimitrios-Emmanuel Spanos, and Nikolas Mitrou. A semantic-based architecture for sensor data fusion. In *2008 The Second International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies*, pages 116–121. IEEE, 2008.
[46] Marco Zappatore, Antonio Longo, Alberto Martella, Benedetta Di Martino, Antonio Esposito, and Salvatore A. Gracco. Semantic models for iot sensing to infer environment–wellness relationships. *Future Generation Computer Systems*, 140:1–17, 2023.

Appendix A. Queries

Query Q_m

```
REGISTER QUERY Traffic AS
    PREFIX ex: <http://myexample.org/>
    PREFIX loc: <https://location.com/>
    PREFIX stat: <https://status.com/>
    PREFIX cnt: <https://cntVehicles/>
    PREFIX concept: <https://concept.com/>

    SELECT ?s
    FROM STREAM <streamIRI_new>
                [RANGE 3s STEP 1s]
    WHERE {
        ?s concept:congestion ?o .
    }
    GROUP BY (?s)
    HAVING (COUNT(?o) > 3);
```

Query 1

```
REGISTER QUERY CongestionDetect AS
PREFIX ex: <http://example.org/>
PREFIX geo: <http://www.w3.org/2003/
            01/geo/wgs84_pos#>
PREFIX stat: <https://status.com/>
PREFIX concept: <https://concept.com/>

CONSTRUCT {
  ?location concept:congestion "congestion" .
}
FROM STREAM <https://www.wtlab.com/TrafficStream>
            [RANGE 3s STEP 1s]
WHERE {
  ?vehicle geo:location ?location ;
           concept:speed ?speedValue ;
           ex:timestamp ?timestamp .
  FILTER(?speed < 50)
}
```

Query 2:

```
REGISTER QUERY CongestionDetect AS
PREFIX ex: <http://example.org/>
PREFIX geo: <http://www.w3.org/2003/01/
             geo/wgs84_pos#>
PREFIX concept: <https://concept.com/>

CONSTRUCT {
  ?location concept:congestion "congestion" .
}
FROM STREAM <https://www.wtlab.com/TrafficStream>
              [RANGE 3s STEP 1s]
WHERE {
  ?vehicle geo:location ?location ;
           concept:speed ?speedValue ;
           ex:timestamp ?timestamp .
  FILTER(?speed < 50)
}
GROUP BY ?location
HAVING (COUNT(?vehicle) > 3)
ORDER BY ASC(?location)
```

Query 3:

```
REGISTER QUERY CongestionDetect AS
PREFIX ex: <http://example.org/>
PREFIX geo: <http://www.w3.org/2003/01/
            geo/wgs84_pos#>
PREFIX stat: <https://status.com/>
PREFIX concept: <https://concept.com/>

CONSTRUCT {
  ?location concept:congestion "congestion" .
  ?location stat:avgSpeed ?avgLocation .
}
FROM STREAM <https://www.wtlab.com/TrafficStream>
            [RANGE 3s STEP 1s]
WHERE {
```

```
  ?vehicle geo:location ?location ;
           concept:speed ?speedValue ;
           ex:timestamp ?timestamp .

  FILTER(?speed < 50)
  FILTER (CONTAINS(str(?location), "2") ||
          CONTAINS(str(?location), "3") ||
          CONTAINS(str(?location), "1"))
}
GROUP BY ?location
HAVING (COUNT(?vehicle) > 1)
BIND(AVG(?speed) AS ?avgLocation)
ORDER BY ASC(?location)
```

Query 4:

```
REGISTER QUERY CongestionDetect AS
PREFIX ex: <http://example.org/>
PREFIX geo: <http://www.w3.org/2003/01/
            geo/wgs84_pos#>
PREFIX stat: <https://status.com/>
PREFIX concept: <https://concept.com/>

CONSTRUCT {
  ?location concept:congestion "congestion" .
  ?location stat:avgSpeed ?avgLocation .
}
FROM STREAM <https://www.wtlab.com/TrafficStream>
            [RANGE 3s STEP 1s]
WHERE {
  ?vehicle geo:location ?location ;
           concept:speed ?speedValue ;
           ex:timestamp ?timestamp .

  FILTER(?speed < 50)
  FILTER (CONTAINS(str(?location), "2") ||
          CONTAINS(str(?location), "3") ||
          CONTAINS(str(?location), "1"))
    UNION
    {
    ?vehicle geo:location ?location ;
                ex:speed ?speed ;
                ex:timestamp ?timestamp .
    FILTER(?speed < 50)
    FILTER (CONTAINS(str(?location), "4"))
    }
}
GROUP BY ?location
HAVING (COUNT(?vehicle) > 1)
BIND(AVG(?speed) AS ?avgLocation)
ORDER BY ASC(?location)
```