



SmartEdge

Semantic Low-code Programming Tools for Edge Intelligence

This project is supported by the European Union's Horizon RIA research and innovation programme under grant agreement No. 101092908

Deliverable D5.1

Design of low-code programming tools for edge intelligence

Lead Editor D. Le-Phuoc (TUB)

Contributors M. Bagheri (CONV), L. Bassbouss (FhG), D Bowden (DELL), P.Cudre-Mauroux (FRIB), K. Dorofeev (SAG), A. Ganbarov (TUB), M. Grassi (FEC), X. Guo (TUB), I. Kosonen (AALTO), A. Le-Tuan (TUB), G. Michelangelo (CNIT), M. Milich (BOSCH), D. Nguyen (TUB), A. Paul (FhG), S. Paul (TUB), E. Petrova (IMC), D. Raggett (W3C), M. Scrocca (CEF), D. Tran (BOSCH), T. Tran (BOSCH), J. Yuan (TUB), N. Zilberman (UOXF)

Version 2.2

Date 21 02, 2024

Distribution PUBLIC (PU)

DISCLAIMER

This document contains information which is proprietary to the SmartEdge (Semantic Low-code Programming Tools for Edge Intelligence) consortium members that is subject to the rights and obligations and to the terms and conditions applicable to the Grant Agreement number 101092908. The action of the SmartEdge consortium members is funded by the European Commission.

Neither this document nor the information contained herein shall be used, copied, duplicated, reproduced, modified, or communicated by any means to any third party, in whole or in parts, except with prior written consent of the SmartEdge consortium members. In such case, an acknowledgement of the authors of the document and all applicable portions of the copyright notice must be clearly referenced. In the event of infringement, the consortium members reserve the right to take any legal action it deems appropriate.

This document reflects only the authors' view and does not necessarily reflect the view of the European Commission. Neither the SmartEdge consortium members as a whole, nor a certain SmartEdge consortium member warrant that the information contained in this document is suitable for use, nor that the use of the information is accurate or free from risk, and accepts no liability for loss or damage suffered by any person using this information.

The information in this document is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information at its sole risk and liability.

REVISION HISTORY

Revision	Date	Responsible	Comment
0.1	01, 06, 2023	TUB	Layout and Structure
0.2	01, 10, 2023	TUB	Initial content
0.3	13, 11, 2023	TUB	Updated layout and structure
0.4	20,11,2023	TUB	Updated content T5.3 and T5.4
0.5	27/11/2023	TUB	New outline for all tasks
0.6	04/13/2023	TUB	Update sections with assignments to partners
0.7	05/02/2023	TUB	Content assignments to all sessions
1.0	12/02/2024	TUB	All contents
1.2	15/02/2024	TUB	First Draft for Task leader reviews
2.0	19/02/2024	TUB	Prepare for final draft
2.2	21/02/2024	TUB	Submitted for Qualify Review

LIST OF AUTHORS

Partner	Name Surname	Contributions
CNIT	Guaitolini Michelangelo	Section 4.4.3
TUB	Danh Le-Phuoc	Executive and outline, Editor of Section 1
TUB	Anh Le-Tuan	Editor of Section 4 and 5
TUB	Duc Manh Nguyen	Section 4 and 5
TUB	Jicheng Yuan	Section 5.4.2.3
TUB	Sumit Paul	Section 4.3.1
TUB	Xuanchi Guo	Section 4.3.2
TUB	Ali Ganbarov	Section 5.4.2.4
BOSCH	Trung-Kien Tran	Editor of Section 3
BOSCH	Duong Tran	Editor of Section 3
BOSCH	Marcel Milich	Editor of Section 3
FRIB	Philippe Cudre-Mauroux	Editor of Section 4
CONV	Mehrdad Bagheri	Section 2.4.2.3, 5.4.1.2.1
CEF	Mario Scrocca	Editor of Section 3.4.3.2
CEF	Marco Grassi	Editor of Section 3.4.3.2
FhG	Louay Bassbouss	Editor Section2.4.2.4; Section 5.4.4.3
FhG	André Paul	Editor Section2.4.2.4; Section 5.4.4.3
DELL	David Bowden	Section 2.4.2.2
AALTO	Iisakki Kosonen	Section 5.4.1.2.1.
SAG	Kirill Dorofeev	Section 5.4.4
W3C	Dave Raggett	Section 2.4.1; 5.4.4.5
UOXF	Noa Zilberman	Section 5.4.4.2
IMC	Elena Petrova	Section 5.4.4.3

GLOSSARY

<i>Acronym</i>	<i>Description</i>
<i>ABR</i>	<i>Adaptive Bitrate Streaming</i>
<i>AMR</i>	<i>Autonomous Mobile Robot</i>
<i>API</i>	<i>Application Program Interface</i>
<i>ARM</i>	<i>Acorn Reduced Instruction Set Machine</i>
<i>CAN bus</i>	<i>Controller Area Network bus</i>
<i>CaS</i>	<i>Compare and Swap</i>
<i>CCTV</i>	<i>Closed Circuit Television</i>
<i>CDN</i>	<i>Content Delivery Network</i>
<i>CNN</i>	<i>Convolutional Neural Network</i>
<i>COCO</i>	<i>Common Object in Context</i>
<i>CPU</i>	<i>Central Processing Unit</i>
<i>CQELS</i>	<i>Continuous Query Evaluation over Linked Streams</i>
<i>CXL</i>	<i>Compute Express Link</i>
<i>DASH</i>	<i>Dynamic Adaptive Bitrate over HTTP</i>
<i>DCAT</i>	<i>Data Catalog vocabulary</i>
<i>DDS</i>	<i>Data Distribution Services</i>
<i>DETR</i>	<i>Detection Transformer</i>
<i>DKG</i>	<i>Dynamic Knowledge Graph</i>
<i>DMA</i>	<i>Direct Memory Access</i>
<i>DPU</i>	<i>Data Processing Units</i>
<i>D-RDMA</i>	<i>Declarative Remote Direct Memory Access</i>
<i>DSL</i>	<i>Domain-specific Language</i>
<i>FaA</i>	<i>Fetch and Add</i>
<i>FAIR</i>	<i>Findability, Accessibility, Interoperability, and Reusability of digital asset</i>
<i>FPGA</i>	<i>Field Programmable Gate Arrays</i>
<i>FRCNN</i>	<i>Fast Region-based Convolutional Neural Network</i>
<i>GDS</i>	<i>Global Data Space</i>
<i>GeoSPARQL</i>	<i>Resource Description Frame geospatial query language</i>
<i>GPM</i>	<i>Graph Pattern Mining</i>
<i>GPS</i>	<i>Global Positioning System</i>
<i>GPU</i>	<i>Graphical Processing Unit</i>
<i>HLS</i>	<i>Hypertext Transfer Protocol Live Streaming</i>
<i>HPC</i>	<i>High-performance Computing</i>
<i>HTTP</i>	<i>Hypertext Transfer Protocol</i>
<i>IDM</i>	<i>Identity Management</i>
<i>IMU</i>	<i>Inertial Measurement Unit</i>
<i>IoT</i>	<i>Internet of Things</i>
<i>IoU</i>	<i>Intersection over Union</i>
<i>IRI</i>	<i>Internationalized Resource Identifier</i>
<i>ISD</i>	<i>Integrated Surface Dataset</i>

<i>iWARP</i>	<i>Internet Wide Area Remote Direct Memory Access Protocol</i>
<i>JPEG</i>	<i>Joint Photographic Experts Group image format</i>
<i>JSON</i>	<i>JavaScript Object Notation</i>
<i>JSON-LD</i>	<i>JavaScript Object Notation Linked Data</i>
<i>JVM</i>	<i>Java Virtual Machine</i>
<i>LAN</i>	<i>Local Area Network</i>
<i>LiDAR</i>	<i>Light Detection and Ranging</i>
<i>MAC</i>	<i>Media Access Control</i>
<i>MAT</i>	<i>Match-action Tables</i>
<i>MAU</i>	<i>Match-action Unit</i>
<i>MEMS</i>	<i>Micro-Electro-Mechanical Systems</i>
<i>ML</i>	<i>Machine Learning</i>
<i>NARF</i>	<i>Normal Aligned Radial Feature</i>
<i>NAS</i>	<i>Neural Architecture Search</i>
<i>NCDC</i>	<i>National Climatic Data Center</i>
<i>NCR</i>	<i>Non-contiguous Regions</i>
<i>NIC</i>	<i>Network Interface Card</i>
<i>OMG</i>	<i>Object Management Group</i>
<i>ONOS</i>	<i>Open Network Open System</i>
<i>OBU</i>	<i>On-board Unit (V2X wireless communication hardware inside a connected vehicle)</i>
<i>P2P</i>	<i>Peer-2-peer</i>
<i>P4</i>	<i>Programming Protocol-independent Packet Processors</i>
<i>PNG</i>	<i>Portable Network Graphics</i>
<i>QET</i>	<i>Query Execution Time</i>
<i>QoS</i>	<i>Quality of Service</i>
<i>QP</i>	<i>Queue Pairs related to Remote Direct Memory Access</i>
<i>QR</i>	<i>Quick Response an evolution of bar codes</i>
<i>RAM</i>	<i>Random Access Memory</i>
<i>RCNN</i>	<i>Region-based Convolutional Neural Network</i>
<i>RTSP</i>	<i>Real-Time Streaming Protocol</i>
<i>RDF</i>	<i>Resource Description Frame</i>
<i>RDMA</i>	<i>Remote Direct Memory Access</i>
<i>REST</i>	<i>Representational State Transfer</i>
<i>RGB</i>	<i>Red, Green, Blue referring to color images</i>
<i>RGBD</i>	<i>Red, Green, Blue, Depth referring to color images with a depth channel</i>
<i>RML</i>	<i>Resource Description Frame Mapping Language</i>
<i>RoCE</i>	<i>Remote Direct Memory Access over Converged Ethernet</i>
<i>ROS</i>	<i>Robot Operating System</i>
<i>RPC</i>	<i>Remote Procedure Call</i>
<i>RSU</i>	<i>Road-side Unit</i>
<i>SGE</i>	<i>Scatter-gather Element</i>
<i>SHACL</i>	<i>Shapes Constraint Language</i>
<i>SLAM</i>	<i>Simultaneous Localization and Mapping</i>

<i>SPARQL</i>	<i>Resource Description Frame query language</i>
<i>TCP</i>	<i>Transmission Control Protocol</i>
<i>TD</i>	<i>Thing Description part of the WoF</i>
<i>TDD</i>	<i>Thing Description Directory part of the WoF</i>
<i>TTL</i>	<i>Time-to-live</i>
<i>UDP</i>	<i>User Datagram Protocol</i>
<i>URI</i>	<i>Uniform Resource Identifiers</i>
<i>UUID</i>	<i>Universally Unique Identifier</i>
<i>V2X</i>	<i>Vehicle to everything</i>
<i>WebRTC</i>	<i>Web Real-Time Communication</i>
<i>WoF</i>	<i>Web of Things</i>
<i>WR</i>	<i>Work Request</i>
<i>YOLO</i>	<i>You Only Look Once</i>

EXECUTIVE SUMMARY

Deliverable D5.1 reports the first iteration of the design of low-code programming tools for Edge Intelligence in the SmartEdge project. The document focuses the design of four parts of the toolchain corresponding to four tasks of WP5: 1) Semantic-driven Multi-modal sensor fusion for edge devices; 2) Swarm Elasticity via Cloud-Edge Interplay; 3) Adaptive Coordination and Optimization; 4) Cross-layer toolchain for Device-Edge-Cloud Continuum. To improve the design's feasibility towards the next implementation stage and KPIs, several initial implementations of individual components or subsystems are carried out to provide insights for their next versions of the implementation in D5.2.

WP5 aims to provide an integrated toolchain to lower the effort in building Edge Intelligence. Based on semantic descriptions of sensing and computing capabilities as well as data queries, the toolchain will decouple the application logic to underlying complicated software, hardware and networking elements. On the other hand, the semantic descriptions and specifications are the key enabler to integrate the elements into execution pipelines at run time without a prior knowledge of them. Hence, the semantic data model is the unified data presentation as the integration point for all components designed in D5.1. Firstly, the sensor fusion of multimodal data of T5.1 will use RDF as the intermediate data representation among the operations that provide the unified input/output data presentations to operators that can be processed and integrated in T5.2, T5.3 and T5.4. Secondly, the declarative programming approach for low-code programming across the layers (e.g. network, RDMA, sensor fusion, orchestration, optimization and runtime) can provide different domain-specific languages (DSLs) that be seamlessly integrated via RDF data model and graph query patterns. In particular, T5.2 allows T5.1 to offload their sensor fusion operations that can be expressed a graph query patterns and similarly T5.3 also can orchestrate the federated processing workloads represented in SPARQL-like query languages. Eventually, the SmartEdge runtime pushes one step further in using RDF data as dynamic knowledge graphs (DKGs) that unifies traditional knowledge graphs and semantic streams. DKGs help to integrate sensory data from T5.1 with operational and environment data, e.g. network telemetries, hardware configuration, training data, host environments, into queryable form with graph query language like SPARQL so that SmartEdge nodes can programmatically access it in a unified via in a distributed fashion.

Table of Contents

1	Introduction.....	1
1.1	Low-code For Edge Intelligence.....	1
1.2	Review of KPIs and Baselines.....	2
2	Semantic-driven Multimodal Stream Fusion For Edge Devices.....	3
2.1	Overview of Stream Fusion.....	3
2.2	Requirements and KPIs.....	4
2.3	Preliminaries and State of the Art.....	6
2.4	Architecture and Design.....	7
2.4.1	Design Overview.....	7
2.4.2	Object Detection, Integration of Sensors, and Scene Understanding.....	8
2.4.3	Semantic Data Stream Fusion.....	24
3	Swarm elasticity via edge-cloud interplay.....	28
3.1	Overview of edge-cloud interplay.....	28
3.2	Requirements and KPIs.....	29
3.2.1	Requirements.....	29
3.2.2	KPIs.....	30
3.3	Preliminaries and state of the art.....	30
3.3.1	An introduction to modern data exchange and its challenges.....	31
3.3.2	Declarative Data Exchange.....	34
3.3.3	Heterogeneous, Low-Code Computing.....	37
3.4	Architecture and design.....	37
3.4.1	General architecture.....	37
3.4.2	Integration with SmartEdge generic architecture.....	38
3.4.3	Integration with Use-Cases.....	38
3.4.4	D-RDMA extensions.....	41
3.4.5	CXL Extensions.....	42
3.4.6	Offloading computations.....	42
3.4.7	Offloading Complex SmartEdge Operations Using SPARQL.....	48
3.4.8	Runtime Optimizer.....	49
4	Adaptive coordination and optimization Mechanisms.....	51
4.1	Overview of Swarm Coordination and Optimization.....	51
4.2	Requirements.....	53
4.3	Preliminaries And State of The Art.....	58
4.3.1	Data Distribution Service -based Communication.....	59

4.3.2	P2P-based Discovery and Federation	70
4.4	Component Design	75
4.5	Semantic-Based Discovery and Formation of Swarm	77
4.5.1	RDfizing P4-based network information into Dynamic Knowledge Graph	79
4.5.2	Network-aware Swarm Formation with DDS.....	82
4.6	Orchestration via Continuous Query Federation.....	86
4.6.1	Preliminary Experiment and Results.....	88
4.6.2	From empirical insights to design and implementation of Orchestrator and Optimizer91	
5	Cross Layer toolchain for device-edge-cloud Continuum.....	93
5.1	Overview of cross-layer tool chain for Device-Edge-Cloud CONTINUUM	93
5.2	Requirements	94
5.3	Preliminaries And Stage of The Art.....	97
5.3.1	Object Detections for Edge Devices	97
5.3.2	Preliminary results for Semantic Programming for edge computing	100
5.4	Model and design.....	101
5.4.1	Semantic Programming Model for Low-code Programming	101
5.4.2	Integrate/deploy/build toolchain into execution target/environments.....	112
5.4.3	SmartEdge Processing Primitives	125
5.4.4	SmartEdge plugins.....	126
5.4.5	Application-support Adaptors/Connectors	133
6	Conclusions.....	141

Table of Figures

FIGURE 1-1. FROM SMARTEDGE LOW-CODE TOOLCHAIN TO SMARTEDGE RUNTIME	1
FIGURE 2-1. ILLUSTRATION OF OBJECT DETECTION, UNIFIED SCENE GRAPH, AND RESPECTIVE SPARQL QUERIES TO OBTAIN INFORMATION.	4
FIGURE 2-2. OVERVIEW OF THE SEMANTIC MULTIMODAL STREAM FUSION PIPELINE.....	7
FIGURE 2-3. AN OVERVIEW OF THE FRAMEWORK FOR SCENE GRAPH GENERATION.....	11
FIGURE 2-4. OBJECT BOUNDING BOXES AND LABELS THAT WERE DETECTED FROM THE CAMERA.	12
FIGURE 2-5. THE EXPLICIT FEATURES OF OBJECTS	13
FIGURE 2-6. A SUBGRAPH OF CONCEPTNET SHOWING THE CONNECTION BETWEEN NODES.....	14
FIGURE 2-7. SENSORS USED IN OPERATIONAL AREA	16
FIGURE 2-8. MEMS LIDAR CAMERAS (LEFT) AND STEREOSCOPIC DEPTH CAMERAS (RIGHT)	16
FIGURE 2-9. RACK WITH QU IDENTIFICATION CODE AND FLOOR MOUNTED QR CODE AND CALIBRATION CHESSBOARD.....	17
FIGURE 2-10. SEMANTIC SENSOR STREAM FUSION PIPELINE	18
FIGURE 2-11. SEMANTIC SCENE GRAPH.....	19
FIGURE 2-12. ILLUSTRATION OF RACK MOVING BETWEEN OPERATIONAL AREAS (LEFT) AND A 2D OCCUPANCY MAP (RIGHT) ..	20
FIGURE 2-13. MEDIA STREAM PROCESSING PIPELINE.....	23
FIGURE 2-14: EXAMPLE OF STREAM DESCRIPTION USING W3C WEB OF THINGS CONCEPTS AND ADDITIONAL METADATA. ..	26
FIGURE 2-15: EXAMPLE OF JSON INPUT AND RDF OUTPUT FOR OBJECT DETECTION.	27
FIGURE 3-1: (LEFT) STANDARD RDMA FORCES THE DATABASE TO ENQUEUE WORK REQUESTS FOR EVERY FRAGMENT	35
FIGURE 3-2: CONTIGUOUS REGIONS ARE INSUFFICIENT TO CAPTURE COMPLEX DATA PATTERNS.	36
FIGURE 3-3. THE LIFE OF AN OPERATION IN THE D-RDMA RUNTIME FROM A SYSTEM'S PERSPECTIVE (A)	37
FIGURE 3-4: SOME OF THE NODES AND HARDWARE DEVICES AVAILABLE IN HELSINKI FOR UC2 ON THE ROAD (A) AND IN INSTRUMENTED CARS (B).	39
FIGURE 3-5: INTEGRATION OF T5.2 WITH UC2; OFFLOADING WILL BE POWERED BY D-RDMA BETWEEN EDGE NODES AND FURTHER EDGE NODES WITH SPECIFIC ACCELERATORS; CLOUD NODES COULD ALSO BE USED IN THAT CONTEXT.	39
FIGURE 3-6: AN INTEGRATED DATA VIEW OF VARIOUS SMART COMPONENTS FOR UC2	40
FIGURE 3-7: AN EXAMPLE OF A D-RDMA EXTENSION TO ACCELERATE FILTERING AND EXCHANGE OF DATA FOR UC2	41
FIGURE 3-8: SAMPLE RESULTS FOR THE OFFLOADED FACE BLURRING OPERATION SHOWING THE ORIGINAL IMAGE.	43
FIGURE 3-9: (TOP) THE SWITCH IS COMPOSED OF A CONTROL PLANE AND A DATA PLANE. THE DATA PLANE HAS A	45
FIGURE 3-10: INTERMEDIATE RESULTS GENERATED FROM GPM COMPUTATION ON TWO WELL-KNOWN GRAPHS	47
FIGURE 3-11: OUR OFFLOADING FRAMEWORK MAIN WORKFLOW. THE EDGE NODES AND A P4 ACCELERATOR	48
FIGURE 3-12: COMPARISON OF RUNNING GPM TASK ENTIRELY ON NODES (SERVERS) USING A STATE-OF-THE-ART.....	48
FIGURE 3-13: THE DESIGN OF OUR RUNTIME OPTIMIZER FOR OFFLOADING OPERATIONS IN SMARTEDGE;.....	49
FIGURE 3-14: THE THREE LAYERS OF OPTIMIZATION THAT WILL BE SUPPORTED BY OUR RUNTIME OPTIMIZER.	50
FIGURE 4-1. ARCHITECTURAL OVERVIEW OF SMARTEDGE ORCHESTRATOR AND OPTIMIZER.....	52
FIGURE 4-2. PUBLISHER FREQUENCY IMPACT ON THE LATENCY USING ECLIPSE CYCLONE DDS SAME DOMAIN WIRELESS COMMUNICATION FOR RASPBERRY Pi3.....	61
FIGURE 4-3. PUBLISHER FREQUENCY IMPACT ON THE LATENCY USING ECLIPSE CYCLONE DDS DIFFERENT DOMAIN WIRELESS COMMUNICATION FOR RASPBERRY Pi3.....	61
FIGURE 4-4. PUBLISHER FREQUENCY IMPACT ON THE LATENCY USING ECLIPSE CYCLONE DDS DIFFERENT DOMAIN WIRED COMMUNICATION FOR RASPBERRY Pi3.....	62
FIGURE 4-5. PUBLISHER FREQUENCY IMPACT ON THE LATENCY USING ECLIPSE CYCLONE DDS DIFFERENT DOMAIN WIRELESS COMMUNICATION FOR RASPBERRY Pi4.....	62
FIGURE 4-6. PUBLISHER FREQUENCY IMPACT ON THE LATENCY USING ECLIPSE CYCLONE DDS DIFFERENT DOMAIN WIRED COMMUNICATION FOR RASPBERRY Pi4.....	63
FIGURE 4-7. PUBLISHER FREQUENCY IMPACT ON THE LATENCY USING ECLIPSE CYCLONE DDS DIFFERENT DOMAIN WIRELESS COMMUNICATION BETWEEN LAPTOP COMPUTERS.....	63
FIGURE 4-8. PUBLISHER FREQUENCY IMPACT ON THE LATENCY USING EPROSIMA DDS DIFFERENT DOMAIN WIRELESS COMMUNICATION BETWEEN Pi3	64
FIGURE 4-9. PUBLISHER FREQUENCY IMPACT ON THE LATENCY USING EPROSIMA DDS DIFFERENT DOMAIN WIRELESS COMMUNICATION BETWEEN Pi4.	65

FIGURE 4-10. PUBLISHER FREQUENCY IMPACT ON THE LATENCY USING EPROSIMA DDS DIFFERENT DOMAIN WIRED COMMUNICATION BETWEEN LAPTOP.....	65
FIGURE 4-11. PUBLISHER FREQUENCY IMPACT ON THE LATENCY USING RTI CONNEXT DDS DIFFERENT DOMAIN WIRELESS COMMUNICATION BETWEEN PI3.	67
FIGURE 4-12. PUBLISHER FREQUENCY IMPACT ON THE LATENCY USING RTI CONNEXT DDS DIFFERENT DOMAIN WIRED COMMUNICATION BETWEEN PI3	67
FIGURE 4-13. PUBLISHER FREQUENCY IMPACT ON THE LATENCY USING RTI CONNEXT DDS DIFFERENT DOMAIN WIRELESS COMMUNICATION BETWEEN PI4	68
FIGURE 4-14. PUBLISHER FREQUENCY IMPACT ON THE LATENCY USING RTI CONNEXT DDS SAME DOMAIN WIRELESS COMMUNICATION BETWEEN LAPTOP.....	68
FIGURE 4-15. ARCHITECTURE OVERVIEW OF P-GRID AND RDF4LED'S INTEGRATION	70
FIGURE 4-16. ATOMIC TRIPLE PATTERN- LISTING ALL OBSERVATIONS	72
FIGURE 4-17. QET OF ATOMIC TRIPLE PATTERN TP1 USING ISD DATASET. N IS THE NUMBER OF PEERS IN THE SYSTEM	73
FIGURE 4-18. QET OF ATOMIC TRIPLE PATTERNS TP1~4 USING ISD DATASET ON 16 PI4s	74
FIGURE 4-19. JOIN QUERY PATTERN CONTAINING 3 ATOMIC TPs -LISTING INFORMATION OF ALL OBSERVATIONS MADE BY A SENSOR	74
FIGURE 4-20. QET OF MULTIPLE JOIN OPERATIONS WITH UNIFORM DATA DISTRIBUTION.....	75
FIGURE 4-21. SYSTEM ARCHITECTURE OF THE COORDINATOR & OPTIMIZER.....	76
FIGURE 4-22. JSON-LD SNAPSHOT OF SEMANTIC DESCRIPTION OF THE CAMARA LOCATED AT JUNCTION 270 IN HELSINKI..	78
FIGURE 4-23. INTEGRATION OF DYNAMIC KNOWLEDGE GRAPH AND P4 RUNTIME FOR PACKET-LEVEL METADATA COLLECTION	80
FIGURE 4-24. RDFIZING SWITCH-SPECIFIC AND ROBOT-SPECIFIC METADATA	81
FIGURE 4-25. SHARED REPOSITORY BETWEEN NETWORK CONTROL PLANE AND MIDDLEWARE LAYER	81
FIGURE 4-26. DYNAMIC KNOWLEDGE GRAPH (KG) USE CASE EXAMPLE WITHIN DDS MESSAGING FRAMEWORK	82
FIGURE 4-27. BUILT-IN TYPES SUPPORTED BY ROS2.....	84
FIGURE 4-28. SEMANTIC ROS DESIGN.....	85
FIGURE 4-29. KEY COMPONENTS TO ENABLE CONTINUOUS QUERY FEDERATION	87
FIGURE 4-30. BASELINE EXPERIMENT RESULT	89
FIGURE 4-31. TOPOLOGY	89
FIGURE 4-32. FAN-OUT FEDERATION EXPERIMENTS: PROCESSING THROUGHPUT RESULTS.....	90
FIGURE 4-33. FAN-OUT FEDERATION EXPERIMENTS: AVERAGE PROCESSING TIME RESULTS	91
FIGURE 5-1 PROFILING YOLO-SERIES MODELS ON JETSON AGX XAVIER.	99
FIGURE 5-2 PROFILING EDGE PERFORMANCE ACCORDING TO DIVER NUMBER OF CAMERAS.	100
FIGURE 5-3 EXAMPLE OF A SEMANTIC PROGRAM WHICH REASONS IF AN OBJECT ENTERS FIELD OF VIEW OF A CAMERA.	102
FIGURE 5-4. EXAMPLE OF DATA SCHEMA TO DESCRIBE RELATIONSHIPS OF CAMERA, OBJECT DETECTIONS AND OBJECT DETECTION RESULTS	103
FIGURE 5-5. EXAMPLE OF A SEMANTIC STREAM SNAPSHOT OF CAMERA FRAME CAPTURED FROM CAMERA 160 DEPLOYED AT JUNCTION 270.....	104
FIGURE 5-6. EXAMPLE SEMANTIC STREAMS PRODUCED BY AN OBJECT DETECTION	105
FIGURE 5-7. CORE CONCEPTS OF SMART TRAFFIC DATA SCHEMA	106
FIGURE 5-8. . A SNAPSHOT OF SEMANTIC DESCRIPTION OF A ROAD SEGMENT CONNECTED TO JUNCTION 270 IN TURTLE FORMAT	106
FIGURE 5-9. SPARQL QUERY TO FIND ALL ROAD SEGMENTS CONNECT TWO JUNCTION USING PROPERTY PATH.....	107
FIGURE 5-10. LANE LAYOUT AT JUNCTION 270 IN HELSINKI SERVES AS AN EXAMPLE.	108
FIGURE 5-11. THE RELATIONSHIPS BETWEEN LANE, JUNCTION AND CONNECTING LANE	108
FIGURE 5-12. OCCUPANCY GRID REPRESENTS A 2-D GRID MAP.....	110
FIGURE 5-13. SEMANTIC SLAM. CAMERA STREAM (LEFT) AND OCCUPANCY GRID MAP (RIGHT) WITH OBJECT DETECTION	111
FIGURE 5-14. SAMPLE SPARQL QUERY FOR PATH PLANING	112
FIGURE 5-15. OVERVIEW OF COMPONENT DESIGN OF SMARTEDGE LOW CODE TOOLCHAIN	113
FIGURE 5-16. EXECUTION CONTEXT ONTOLOGY EXTENDED FROM SMARTEDGE SCHEMA FROM D3.1	114
FIGURE 5-17. COMPONENT DESIGN OR SMARTEDGE RUNTIME	115
FIGURE 5-18 OVERVIEW OF VISIONKG PLATFORM	117

FIGURE 5-19 DATA QUERYING AND PIPELINE CONSTRUCTING VIA VISIONKG.....	118
FIGURE 5-20 DATA DIVERSITY IN VISIONKG	119
FIGURE 5-21 STATISTICS OF TRIPLES AND ENTITIES IN VISIONKG FOR OBJECT DETECTION AND IMAGE CLASSIFICATION DATASETS.....	120
FIGURE 5-22. VECTOR SEARCH TIME COMPARISON BETWEEN TORNADO AND OPENCL	122
FIGURE 5-23. DISTRIBUTED EXECUTION ON ABSTRACTED HARDWARES	123
FIGURE 5-24. ARCHITECTURE MODULES OF A SWARM APPLICATION WITH ONE SMART-NODE AND ONE COORDINATOR	127
FIGURE 5-25. ARCHITECTURE OF DISTRIBUTED IN-NETWORK COMPUTING FRAMEWORK	128
FIGURE 5-26. COGNITIVE ARCHITECTURE FOR CHUNKS & RULES.....	131
FIGURE 5-27. CHUNKS & RULES SYNTAX AS RAILROAD DIAGRAMS	132
FIGURE 5-28. METRIC REPORTING AND VISUALIZATION.....	134
FIGURE 5-29. METRICS DASHBOARD.....	135
FIGURE 5-30. REMOTE RENDERING AND STREAMING ARCHITECTURE	137
FIGURE 5-31. ARCHITECTURE OF MULTI-VIEW VISUALIZER TOOL	137
FIGURE 5-32. MULTI-VIEW VISUALIZER SCREENSHOT.....	138
FIGURE 5-33. SIMSWARM – SCREENSHOT OF SMART WAREHOUSE DEMO WITH ROBOT FORKLIFTS,	139

1 INTRODUCTION

D5.1 proposes the design of the low-programming tools for end intelligence which constitutes the designs of following tools: i) semantic-based multimodal sensor fusion for edge devices (Section 2 for T5.1), ii) Swarm elasticity via edge-cloud interplay (Section 3 for T5.2); iii) Adaptive Coordinator and Optimization (Section 4 for T5.3); iv) Cross-layer toolchain for device-edge-cloud continuum (Section 5 for T5.4). The designs all of such tools employ the same approach for low-code programming, declarative programming (cf Section 1.1). The artifacts of such tools along with the ones produced in WP3 and WP4 will be integrated and deployed as a runtime for a SmartEdge Swarm node as illustrated in following Figure 1-1.

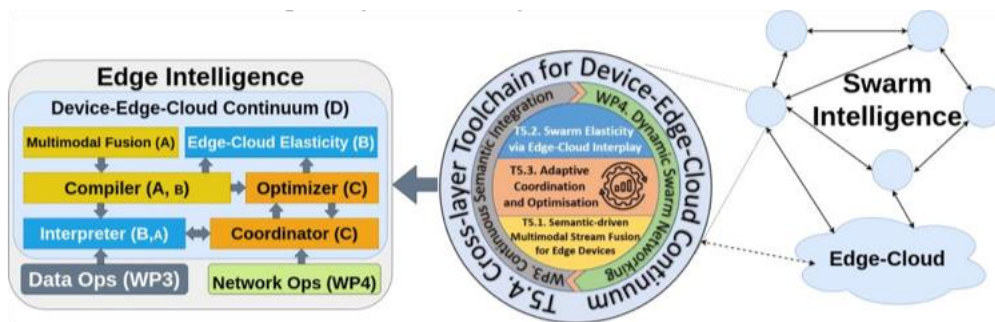


Figure 1-1. From SmartEdge low-code toolchain to SmartEdge runtime

1.1 LOW-CODE FOR EDGE INTELLIGENCE

The hallmark of low-code programming of SmartEdge lies in its declarative programming model, an approach that focuses on ‘what’ a domain expert wants a program should behave not ‘how’ to integrate constituent components together to build such a program. At its core, this model uses semantic data model, i.e. RDF, to interlink domain-expert knowledge with input data and necessary components. Hence, it enables the creation of a toolchain running on edge-cloud execution environments by leveraging the domain-specific knowledge of experts, organized systematically within dynamic knowledge graphs (DKGs). These DKGs serve as the foundation for constructing instructions that edge devices can execute or interpret, thus materializing expert insight into computable operations.

The domain experts in SmartEdge are UC owners, traffic, car or robot engineers who will declaratively the ‘what’ a swarm of edge nodes should do via Domain-Specific Languages (DSLs) that embody semantics familiar to their application domains, e.g. traffic or factory floor, called Semantic DSLs. Such DSLs are constructed from standardized data models and ontologies, e.g. RDF, SPARQL, SHACL and Recipes (cf D3.1). By reducing the complexity of traditional programming constructs, e.g. C++ or python, Semantic DSLs empower domain experts to directly contribute to the development process, thus mitigating the necessity for in-depth programming expertise in Data Ops, Network Ops and AIOps.

Furthermore, Smart low-code tools are not solely focused on the high-level application design; they also strive to enhance the productivity of developers working at the lower layers of the technology stack of Edge Intelligence, e.g. such as networking programming, Remote Direct Memory Access (RDMA), and C++/python for ROS. By automating the time-consuming and monotonous tasks traditionally performed manually, low-code toolchain aims to reduce the potential for error and the overall burden on developers.

The strategic advantage of low-code toolchain of WP5 is manifold. It seeks to alleviate the shortage of developers skilled in lower-level programming languages such as C/C++ and P4. By simplifying complex coding tasks and reusing coding artifacts in WP3-WP5, it minimizes the likelihood of mistakes associated with manual coding and amplifies the efficiency gains across an organization, enabling one individual's low-code program to benefit their colleagues.

In essence, the SmartEdge low-code toolchain synergizes the intricate knowledge of subject matter experts with the operational logic of computational systems. The seamless integration of Semantic DSLs within the low-code paradigm exemplifies a shift towards a more inclusive and efficient software development lifecycle, one where domain expertise is harnessed to its fullest potential, fostering a collaborative and productive environment for developers across the layers of swarm intelligence.

1.2 REVIEW OF KPIS AND BASELINES

D5.1 reviewed five KPIS and then established the baselines during the design phase to give empirical insights as concrete references of following-up implementation and validation steps.

ID	Descriptions	Baselines
K4.1	Ability to free developers from specifying capabilities of hardware and sensors at the design phase of stream fusion pipelines with end-to-end latency guarantee (e.g., 20-75% lower latency to baselines [DTH+21, PET21])	Initial baselines [DTH+21, PET21] will be evaluated in edge devices and compared with solutions proposed in Sect. 5.3).
K4.2	Ability to elastically scale 200-500% better than the state of the art, e.g., [Cudre-Mauroux13, Duc21, Schneider22].	Specific baselines will be selected for each application, such as [Schneider22] for graph applications, [Cudre-Mauroux13] for distributed RDF processing or [Duc21] for stream fusion.
K4.3	Can dynamically optimize resource to be 50%-150% better in terms of computing resources and bandwidths;	Initial baselines implemented and profiled at middleware level(Sect. 4.3.1.14.3.14.3.1.3), P2P storage level(Sect.4.3.2 4.3.2.2) and query federation level(Sect. 4.6.14.6.1)
K4.4	Lower the effort in building swarm intelligence with the target of reducing the coding effort in comparison to imperative programming paradigms by 80-90%, e.g., Python or C++, with SMARTEDGE low-code tool chain.	Off-the-self implementations in Python for UC2, UC3 and UC4 will be used as baselines for low-code programming workflow of T5.4
K4.5	Support AI operations and coordination on a large number of heterogeneous IoT devices (20-50 types of 200-1000 devices) and smart systems (5-10 application domains) to achieve a higher resilience in terms of being able to integrate new sensors and participant nodes at runtime without interrupting the current application logic	Initial baselines are developed and profiled in Section 6.3 towards realistic data collected from UC2 and UC3, then, test to scale of 200-300 devices (e.g. device types listed in Sect.5.3.1)

2 SEMANTIC-DRIVEN MULTIMODAL STREAM FUSION FOR EDGE DEVICES

2.1 OVERVIEW OF STREAM FUSION

In this section, we present the overview of this task and its connection to the other tasks and use cases in the project.

The task will develop components to fuse multimodal stream data into high-level information to be interoperable at semantic levels defined in WP3. The components will be parts of the toolchain in WP5 to provide low-code access, e.g via SPARQL queries, to information about the (swarm) devices and their surrounding environment. More specifically, given video frames and sensor devices as inputs, we will provide components for the following subtasks:

1. Performing object detection and/or semantic segmentation to recognize devices and their relations to each other and to the environment.
2. Alignment with ontology schemas developed in WP3.
3. Generating a semantic scene from video frames.
4. Integrating other sources of information e.g., sensor data.
5. Fusing these multimodal data so that it can be streamed, e.g. as RDF data, to use cases.

Figure 2-1 illustrates an initial step for the process of creating a semantic scene graph that represent multiple modal information. We first perform object detection and extract their annotation labels. To create a unified data model for the annotation labels and visual features, we follow the Linked Data principles and use the RDF data model. The data entities (e.g., images, boxes, labels) are named using Uniform Resource Identifiers (URI). RDF data model allows the data to be expressed using triples of the form $\langle \text{subject}, \text{predicate}, \text{object} \rangle$. For example, to describe “an image (a video frame) contains a bounding box for a person”, we first assign unique URIs, e.g., `smartedge/img01` and `smartedge/box01`, for the image and the bounding box, respectively to create the following triples for such image: $\langle \text{img01}, \text{hasBox}, \text{box01} \rangle$, $\langle \text{box01}, \text{hasObject}, \text{obj01} \rangle$, $\langle \text{obj01}, \text{rdf : type}, \text{Person} \rangle$; for simplicity, we skip the prefix in these triples. Furthermore, we add metadata and semantic annotations, e.g. RDF-Star or use a reification technique. The created semantic scene graph will facilitate semantic reasoning capability and the interoperability over the whole project. It also enables for integrating multimodal information, e.g. coming from semantic SLAM (see also Sec 5.4.1.2.2) or external predefined domain knowledge.

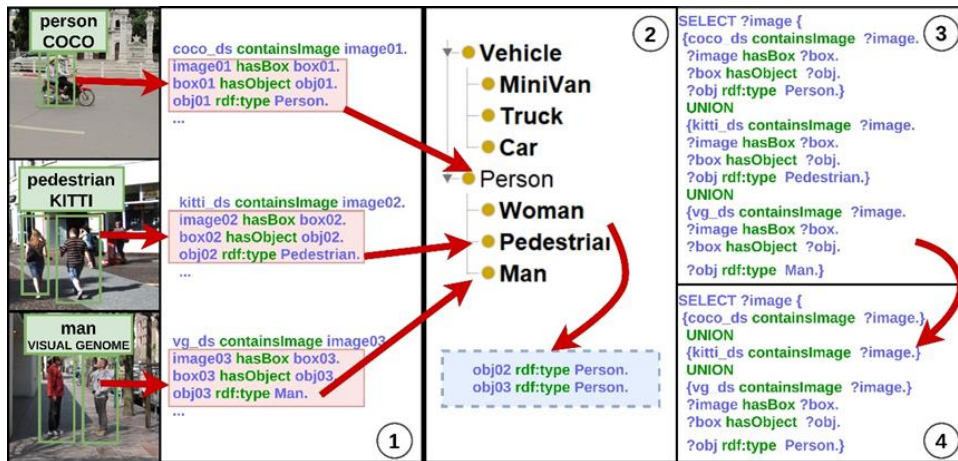


Figure 2-1 Illustration of object detection, unified scene graph, and respective SPARQL queries to obtain information.

We will collaborate with tasks in WP3 to define semantics of the data, e.g., via ontologies, and the data model to represent multimodal information. This task also requires collaboration with Task T5.2 when the computation needs to be shifted to nodes that are more powerful than edge devices.

2.2 REQUIREMENTS AND KPIS

ID/Ver: SW-012/v1.2	Related Use Case(s): UC-3	Task: T5.1
Selected data originated by one node may be provided to multiple swarms. For example, when a node is providing a sensor stream that could be of use to multiple swarms.		
This requirement is addressed by using SPARQL queries specifying node and specific data used in UC-3		

ID/Ver: IDM-011/v1.2	Related Use Case(s): UC-1, UC-2, UC-3, UC5	Task: T5.1
Expression of situation specific conditions that trigger obfuscation or blurring of images and mechanisms to execute.		
This situation will be described using RDF(S).		

ID/Ver: IDM-015/v1.2	Related Use Case(s): UC-3	Task: T5.1
Consent management system and compliance with metadata.		
Task 5.1 will provide semantic description of all the data sources including metadata (Sec 2.4.1, 2.4.2)		

ID/Ver: IDM-016/v1.1	Related Use Case(s): UC-3	Task: T5.1
Methods for aggregation and differential privacy applied to streams.		
Task 5.1 will implement this mechanism, e.g. via context attention outlined in Sec. 2.4.1.		

ID/Ver: LC-019/v1.2	Related Use Case(s): UC-2, UC-3	Task: T5.1
It should be possible to easily execute semantic queries, e.g. using scene understanding and querying scene graphs, over sensor data (e.g., queue lengths in traffic situations) and make that information available for the SmartEdge swarm nodes.		
Task 5.1 provides graph query engine, e.g. SPARQL query engine, so that swam nodes can execute semantic queries and obtain necessary information (Sec.2.4.2, 2.4.3)		

ID/Ver: AL-010/v1.2	Related Use Case(s): UC-3	Task: T5.1
A tool must be implemented to take the image feeds from multiple cameras and triangulate the position and pose of the object based on the image, e.g., three images could be obtained by the three overhead cameras showing the same object from three different views. By combining the three images, the tool should be able to orient the object in 3D space.		
Task 5.1 provides component to construct semantic scene graphs (Sec 2.4.1, 2.4.2), which provide details of objects in multiple cameras. It might be useful for 3D object orientation but does not directly provide this function.		

ID/Ver: AL-011/v1.1	Related Use Case(s): UC-2, UC-3, UC-5	Task: T5.1
A tool must be implemented to classify known objects in the specific use case domain, but should also support the “unknown” classification, i.e., an object has been detected, but can’t be classified with a reasonable degree of certainty.		
Task 5.1 (Sec. 2.4.2) provides classification models in which “unknown” category is considered.		

ID/Ver: CSI-003/v1.1	Related Use Case(s): UC-1, UC-3	Task: T5.1
SmartEdge must provide tools or components for semantic segmentation of the objects of interest in the environment. For examples, vehicles in traffic or AMRs in factories.		
Task 5.1 (Sec 2.4.1, 2.4.2) provides semantic scene graphs for each application domain. This scene graphs contain semantic segmentation of objects and other information in the environment.		

ID/Ver: CSI-004/v1.1	Related Use Case(s): UC-1, UC-3	Task: T5.1
SmartEdge must provide components for semantic scene construction. This is the follow up step for CSI-003.		
Task 5.1 (2.4.1, 2.4.2) provides semantic scene graphs. This is the main objective of Task 5.1.		

ID/Ver: CSI-005/v1.2	Related Use Case(s): UC-1, UC-3, UC-5	Task: T5.1
SmartEdge must provide mechanisms to formalize external knowledge, (e.g., traffic rules or physiotherapists’ rules, therapies, tasks), that are applicable for the current scene.		
This requirement is addressed together with WP3 (Task 3.1) . In principle, Task T5.1 enables the integration of rule-based languages, e.g. SHACL, RDFS, or OWL rules.		

2.3 PRELIMINARIES AND STATE OF THE ART

One of the most critical challenges arising in many use cases within and without SmartEdge is how to combine multiple data types into a uniform representation. This merging process is called data stream fusion. According to Gao et al. [Gao20], data fusion aims to “integrate the data of different distributions, sources, and types into a global space in which both inter-modality and cross-modality can be represented in a uniform manner”. This is often achieved by leveraging the information each individual modality provides, as well as the cross-modality interactions

between data types. We extend this notation to semantic data stream fusion by adding semantics level to the data and integrating them in the streaming manner.

Stream fusion is the cornerstone of other features and use cases within SmartEdge that allows it to integrate information from more than one modality. We first present the overview of popular data stream fusion mechanisms and then propose the mechanism adopted for SmartEdge. Those mechanisms differ based on when they occur in the event detection pipeline: (i) data characterization-based approaches operate at the feature level, (ii) transformation-based approaches learn the encoding of the features and fuse them at the representation level, or (iii) at the decision level, by aggregating detection scores.

Data Characterization-Based Fusion. Fusion based on data characterization is implemented by extracting key features from each modality and combining them into a joint representation. Data characterization-based fusion mechanisms are composed of three stages. First, features are manually extracted from each modality. Second, the selected features are preprocessed, by applying normalization, scaling them to the same magnitude, or projecting them into the same vector space. Third, the modalities are combined based on the extracted characteristics into a fused data format.

Since the features extracted from the input modalities are usually of different scales, several techniques apply rudimentary preprocessing such as normalization. Other proposed fusion mechanisms to compute similarity scores within each modality such as geographic distances, keyword overlaps, etc. To combine the preprocessed features, some techniques opt for a simple concatenation of the values into a fused feature vector. Alternatively, one can combine the features extracted from multimodal data source into a graph representation.

Transformation-Based Fusion. Transformation-based data fusion mechanisms include a representation learning component, which is first trained on labeled data to transform each modality into an internal latent representation. The data fusion occurs only in the second step, by combining the transformed representations. The representation learning is achieved using neural network architectures, with distinct models for each modality. These models transform the input data into latent representations, which are then fused. For instance, the authors in [Chen21, Khadanga19] transform time series and text by training two deep neural networks to produce distinct latent vectors. These transformed representations are then fused, either by simple concatenation or by means of an additional fully-connected neural network layer [Yang 21]. This type of architecture allows for an end-to-end pipeline, integrating the transformation-based fusion with the downstream event detection task. During the training phase of the neural networks, a backpropagation step iteratively improves the learned transformation functions.

Decision-based fusion. Decision-based fusion is usually used in a specific task or use case. It consists of fusing separated scores, which reflects the goal of that use case, obtained from each modality separately. To combine the results at the decision level, data fusion mechanisms

compute averages of the individual scores or train a model taking as input the individual scores and producing a fused result. Combining distinct detection scores can be done by simply averaging these scores or by weighted sum of all scores.

In SmartEdge, different use cases have different characteristics and requirements. To keep our solutions generic enough, we propose to have a unified semantic representation of different information sources and optionally feature or vector representation of distinct sources that we will describe in the next section.

2.4 ARCHITECTURE AND DESIGN

2.4.1 Design Overview

In this section, we present our design of T5.1 for multimodal semantic stream fusion of SmartEdge. Figure describes the overview of the semantic multimodal stream fusion pipeline. This pipeline takes the media data, e.g. camera frames, and sensor data as inputs and provides a semantic description of those data to down-stream tasks in different use cases.

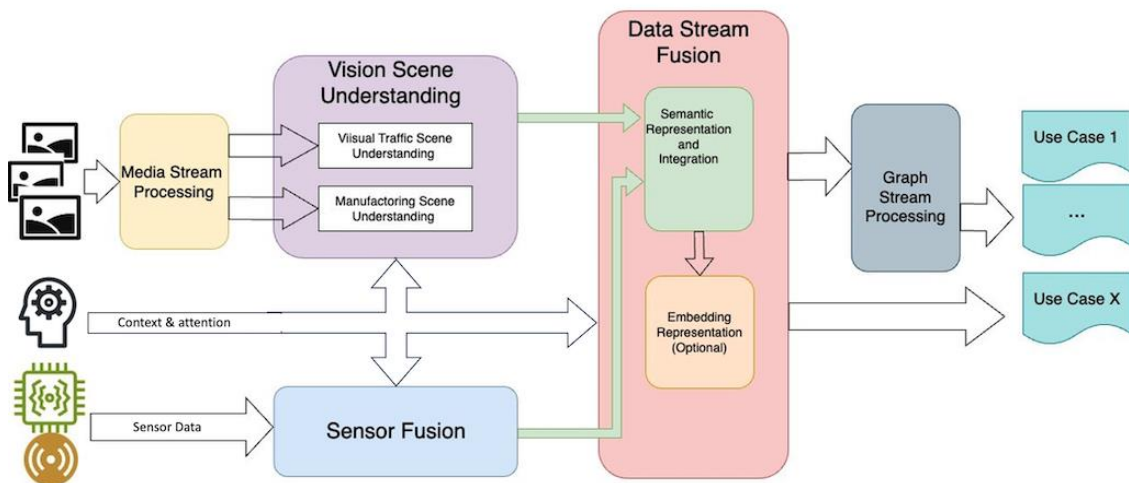


Figure 2-2. Overview of the semantic multimodal stream fusion pipeline

Context & Attention. In addition to input data, applications should be able to specify the context for interpreting sensor data. In essence, the context determines which features are considered to be important to the application, e.g. in an automotive context, we are interested in vehicles, pedestrians, lane markings, road signs and traffic lights, etc. In some cases, the context is predetermined and not subject to change. In other cases, the context depends on the application goals that vary over time. The context makes it possible to efficiently identify relevant features in cluttered sensory environments, allowing the processor to ignore features irrelevant to the current application needs.

Applications should also be able to specify the focus of attention to direct processing resources to the specific features of interest at a given moment in time. An example might be a specific vehicle in the field of view, or a specific road sign when we need to understand what it signifies, e.g. information on which lane to take when crossing a junction. The means to specify context and focus shows the need for backward flows that complement the feedforward flows involved in progressively higher levels of interpretation of sensor data.

A further architectural capability is the means to draw the application's attention to features that require immediate attention, for example, when a child runs out into the road after a ball. This can be modelled as events that will trigger urgent behavioural changes. The classes of events of interest can be described as part of the context model.

The Media Stream Processing Component (Section 2.4.3.2). This component includes the software components required to make media streams from capturing devices for the different Use Cases like 2D Cameras, LiDAR Cameras, etc. from cars or robots available in the appropriate format for further components like Data Fusion.

Sensor Fusion Component (Section 2.4.3.1). This component aggregates the sensor data, e.g. to calculate *traffic indicators* (queue length, vehicle density, option zone metrics, etc.) and for the other prediction tasks.

Vision Scene Understanding (Section 2.4.2.1-2). The main goal of this component is to perform vision tasks to identify objects in the scene, their relations between each other, and details of the surrounding environment. There are two sub-components dedicated to traffic scene and manufacturing scene.

Data Stream Fusion (Section 2.4.3). This component integrates all form of information, adding semantic descriptions using a unified ontology/schema and semantic web standards, e.g. RDF(S). Optionally, embedding representation of the data might be provided, depending on the requirements from use cases.

Graph Stream Processing (Section 5.4.3). This component takes the semantic representation from the previous component, performs query planning and execution, and provides the results/information to the use cases.

2.4.2 Object Detection, Integration of Sensors, and Scene Understanding

2.4.2.1 Scene Understanding in Traffic

2.4.2.1.1 Introduction

In this section, we detail the component "Visual Scene Understanding" in the architecture outlined in Figure 2-2.

Taking inspiration from the innate human capacity to effortlessly interpret and comprehend visual scenes, visual scene understanding has long been hailed as the goal in computer vision. It has already garnered significant interest from the research community. From the point of view of graph theory, a scene graph is a directed graph with three types of nodes: object, attribute, and relation. However, for the convenience of semantic expression, a node of a scene graph is seen as either a "subject" or an "object" with all its attributes, while the relation is called a "predicate", which describes how the "object" is related to the "subject". A subgraph can be formed by connecting multiple triplets, containing all the adjacent nodes of the object. Therefore, these adjacent nodes directly reflect the context information of the object. From the top-down view, a scene graph can be broken down into several subgraphs, a subgraph can be split into several triplets, and a triplet can be split into individual subject, object with their attributes and relation. Accordingly, we can find a region in the scene corresponding to the substructure that is a subgraph, a triplet, or an object. From this strict correspondence, a conclusion can be drawn that the scene graph corresponding to a given scene is structurally

unique without considering differences in the semantic expression in a dataset with definite relation categories and object classes, though, most of the time, it is incomplete. The uniqueness supports the argument that the use of a scene graph as a replacement for a visual scene at the language level is reasonable.

The general approach for scene graph generation typically involves a two-stage process. In the initial stage, Object Detection is performed, wherein the goal is to identify and categorize objects within a given visual scene. This task encompasses both localizing the objects and assigning them to specific categories. Following Object Detection, the second stage focuses on Visual Relationship Detection (VRD). In this stage, the model classifies the relationships between pairs of detected objects, providing a more holistic understanding of the scene beyond individual object instances. This two-stage approach allows for the creation of comprehensive scene graphs that represent both the objects present and the relationships between them, contributing to a richer interpretation of visual content in computer vision applications.

2.4.2.1.2 Related work and state-of-the-art

Object Detection

The ground-breaking R-CNN families, including fast R-CNN [Girshick15], faster R-CNN [RenShaoqing15], R-FCN [Dai16], and Libra R-CNN [Pang19], alongside various iterations such as ATSS [Biffi20], RetinaNet [Lin17], and FCOS [Tian19], have significantly advanced the field of object detection. Central to their design is the one-to-many label assignment strategy, wherein each actual object box is allocated to multiple coordinates in the detector's output. This approach is employed in conjunction with proposals, anchors, or window centers as supervised targets. Despite their impressive performance, these detectors heavily rely on several manually crafted elements, such as non-maximum suppression procedures or anchor generation. In order to create a more adaptable end-to-end detector, the DEtection TRansformer (DETR) [Carion20] was introduced, conceptualizing object detection as a set prediction challenge. It incorporates a one-to-one set matching scheme within a transformer encoder-decoder architecture. Consequently, each actual object box is exclusively assigned to a specific query, eliminating the necessity for multiple manually designed components that encode prior knowledge. This strategy introduces a versatile detection pipeline, paving the way for numerous DETR variants aimed at enhancing its performance further.

The pioneering object detector, DETR [Carion20], which relies on the transformer architecture, integrates a one-to-one set matching approach for object detection, enabling fully end-to-end object detection. The one-to-one set matching method initially computes the overall matching cost using Hungarian matching and assigns a single positive sample with the lowest matching cost to each actual object box. However, DNDETR demonstrates the slow convergence results attributed to the instability of one-to-one set matching, prompting the introduction of denoising training to address this challenge. YOLOv4 (You Only Look One-level [Chen21]) prioritizes the swift operational speed of an object detector in production systems and emphasizes optimization for parallel computations, as opposed to focusing on the theoretical indicator of low computational volume (BFLOP). DINO [Zhang22] builds upon the sophisticated query formulation from DAB-DETR [Liu22] and integrates an enhanced contrastive denoising technique to attain cutting-edge performance. Group-DETR [Chen23] introduces a group-wise one-to-many label assignment to leverage multiple positive object queries, akin to the hybrid matching scheme in H-DETR [Jia23]. Co-DETR [Zong23], another DETR-based detector, addresses the inherent limitation of one-to-one set matching, which tends to cause significant

inefficiencies during training. Consequently, this approach significantly alleviates the challenges associated with the encoder's feature learning in one-to-one set matching.

Visual Scene Graph Generation

In the last ten years, there has been significant interest in scene graph generation (SGG) across different communities. This interest stems from SGG's capacity to accurately depict the semantic aspects of complex visual scenes. The primary goal of SGG is to recognize all objects present in an image along with their visual relationships, requiring a fusion of visual perception and natural language understanding. Consequently, substantial efforts have been dedicated to aligning visual and semantic spaces to facilitate relationship representation learning. Additional investigations have emphasized the significance of each subject-object pair in deducing ambiguous relationships. Xu et al. [Xu17] suggest an iterative refinement approach by conveying contextual messages, while Zellers et al. [Zellers18] employ a bidirectional LSTM to capture a global context. Their notable success underscores the essential role of spatial context in recognizing visual relationships. Consequently, subsequent studies emphasize the importance of spatial context in the creation of scene graphs. Many of these works leverage graph convolutional networks or similar architectures to facilitate message passing among diverse objects. Chen et al. [Chen19] construct a graph that links identified objects based on statistical correlations, utilizing it to comprehend context among different objects for prediction regularization. In a similar vein, Tang et al. [TangKaihua19] devise a dynamic tree structure for efficient encoding of context among various object regions. As the transformer model continues to make remarkable advancements, an increasing number of studies suggest employing this model type to acquire more representative features from spatial context. Cong et al. [Cong23] introduce an encoder-decoder architecture that engages various attention mechanisms, coupled with subject and object queries, to reason about visual feature context and relationships. Kundu et al. [Kundu23] propose contextualized relational reasoning through a two-stage transformer-based architecture, facilitating effective reasoning over cluttered and intricate semantic structures. While these models exhibit impressive progress in static images, they may encounter substantial performance declines when applied to discern dynamic visual relationships in videos. This is due to the necessity for thorough exploration of temporal consistency and transition correlations across different frames.

Visual Relation Detection

Hong and Pavlic [Hong21] introduce a zero-shot learning approach utilizing the Randomly Weighted Feature Network (RWFN). This approach leverages similarities with other seen relationships and background knowledge, represented through logical constraints between subjects, relations, and objects. The goal is to predict triples that are absent from the training set. Cui and Farazi [Cui22] propose VReBERT, a BERT-like transformer model designed for Visual Relationship Detection. It incorporates a multi-stage training strategy to jointly process visual and semantic features. Jiang and Taylor [Jiang23] present a discovery that exploiting hierarchical structures among labels for relationships and objects significantly enhances the performance of scene graph generation systems. They demonstrated that incorporating hierarchical relationship reasoning can substantially improve the performance of a baseline model.

2.4.2.1.3 Proposed solutions

The basic framework for scene graph generation is shown in Figure . Given a video frame, the goal of the system is to generate a directed graph that strictly reflects the semantic relationship between objects within the scene. Using an off-the-shelf object detector, the system first extracts explicit features for objects, specifically their visual features, bounding boxes, and labels. The labels are then used for mapping objects to nodes in a knowledge base. A knowledge base, such as ConceptNet [Speer17], DBpedia [AuerSören17], Wikidata [Vrandečić14], is an enormous set of triplets, where a triplet is in the form of (node, edge, node). This structure has the same idea with a scene graph, which is also a set of triplets in the form of (subject, predicate, object). Thus, the relations between nodes from a knowledge graph can be considered the implicit features of the objects. These explicit features and implicit features can be combined and further refined. Finally, predicate classifiers are used to predict the categories of objects and predicates, and the scene graph is generated.

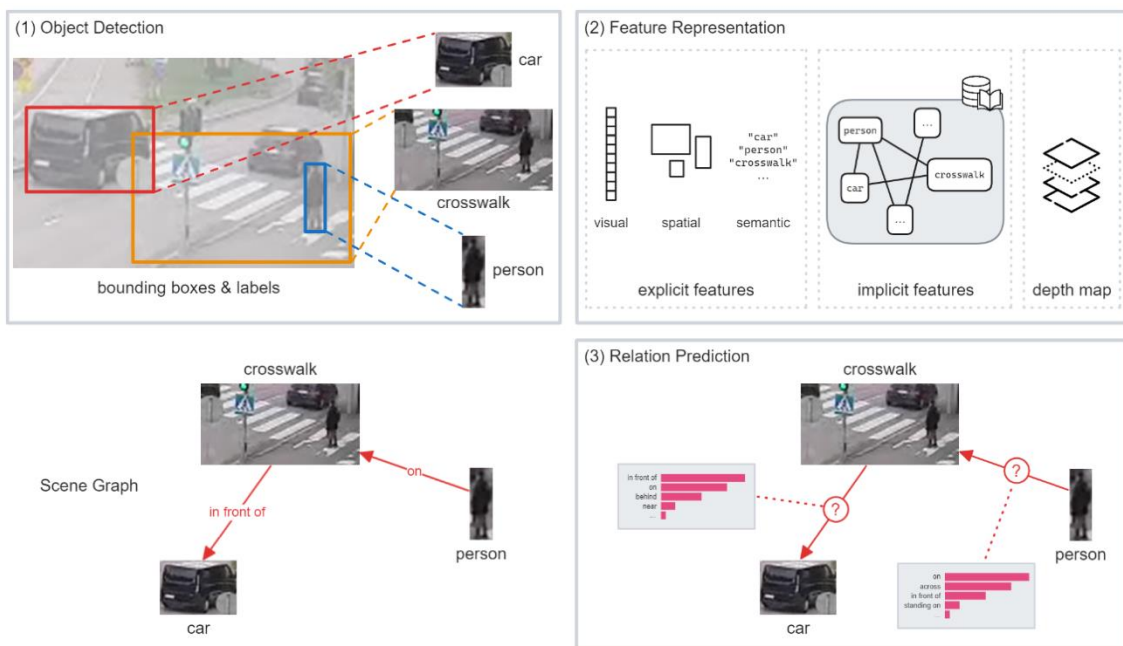


Figure 2-3. An overview of the framework for scene graph generation

Object Detection

At the forefront of the visual relationship detection process, the object detection module serves as the initial component in the pipeline. In this project, we employ Grounding DINO [LiuShilong23], a robust open-set object detector designed to identify diverse objects specified through human language inputs. The objective is to create a comprehensive scene graph by detecting all potential objects or object proposals within the image, grouping them into pairs, and utilizing the features of their combined area (referred to as relation features) as the fundamental representation for predicate inference. Figure illustrates an instance of object bounding boxes successfully detected from a real-time video. The prompt texts for the detector were "person", "crosswalk", "car", and "traffic light".

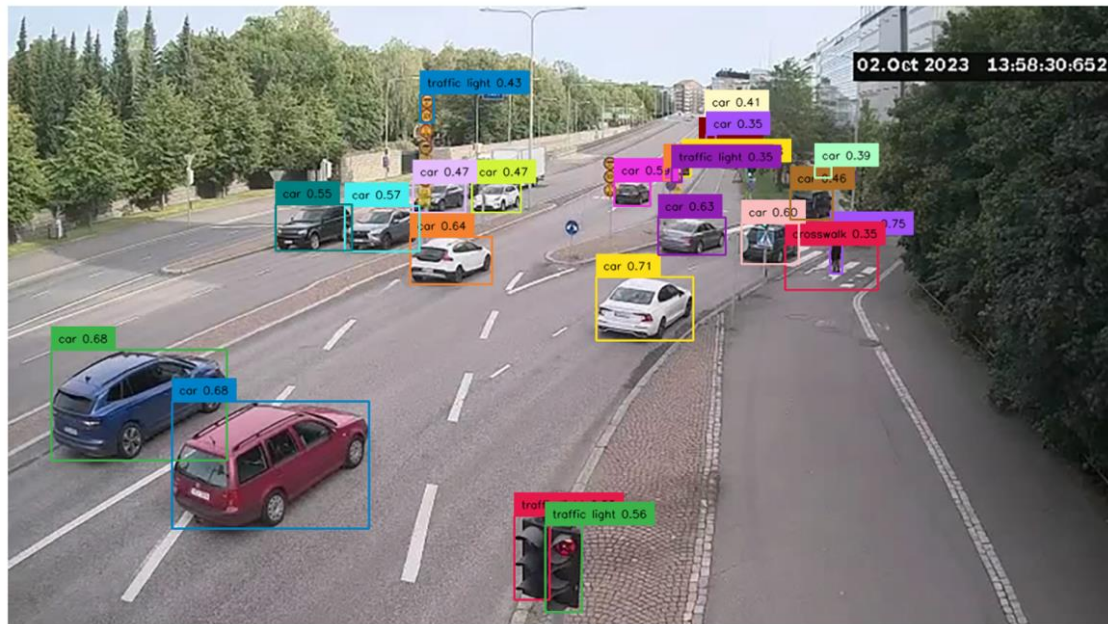


Figure 2-4. Object bounding boxes and labels that were detected from the camera.

Feature Representation: Explicit Features

From the objects that were detected by the object detector, we construct the explicit features for the relationship of each pair of objects in three aspects: visual, spatial, and semantic. Visual features are the CNN features of the two objects. Spatial features are the bounding boxes and coordinates of the two objects which encode their spatial layouts. Semantic features are the class labels of the two objects that provide a strong prior of the predicate. It is a very popular approach that these three features are combined in an early stage to learn a compositional feature for relationship prediction. However, that makes the contribution of each feature unclear and unoptimized. Inspired by the work of Zhang et. al [ZhangJi18], we separate the three features in an explicit and interpretable way. We further assume that the three features are independent from each other. So we can build three separate branches of sub-networks for them. How we optimize each of these features will be investigated later. We only fuse the optimized features in the final stage to get the relation prediction. The idea of explicit features is illustrated in Figure .

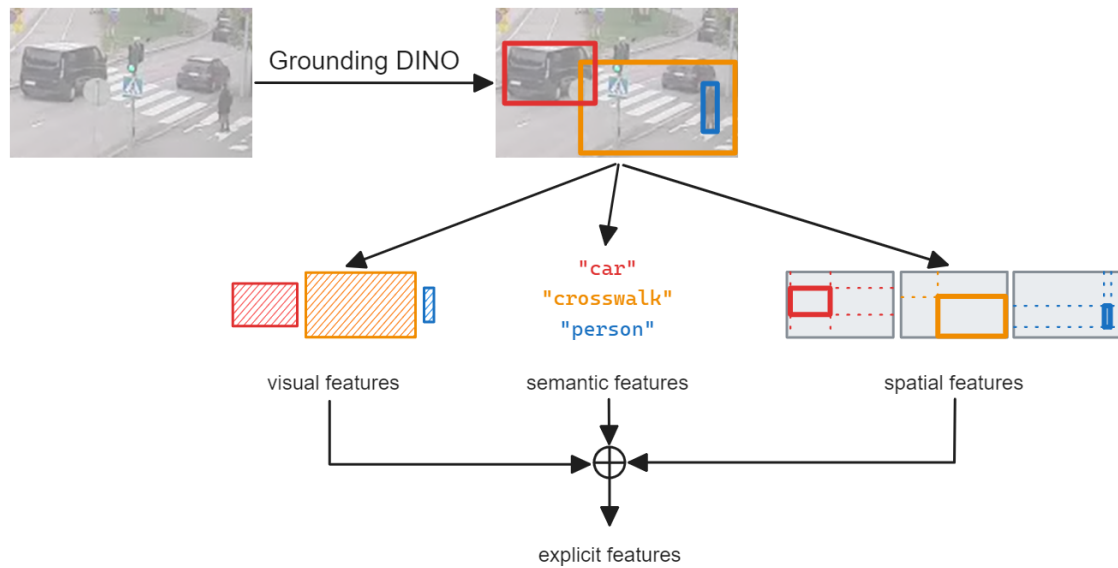


Figure 2-5. The explicit features of objects

Feature Representation: Implicit Features

The scene graph is a semantically structured description of a visual world. Although visual relationships are scene-specific, there are strong semantic dependencies between the scene entities (subject and object) and the relationship predicate in a relationship triplet (subject, predicate, object). For example, when "person", "car" and "crosswalk" are detected individually in an image, the most likely visual triplets are (person, on, crosswalk). Similarly, (car, on, crosswalk) is possible, though less frequent. But (car, on, person) is normally unreasonable. We can observe that the determination of the relation category often depends on the labels of the participating subject and object. This allows us to assign weights to the probability output of relationship detection networks based on statistical co-occurrences, potentially enhancing the performance of visual relationship detection. Therefore, we suggest leveraging ConceptNet [Speer17], a popular source for commonsense knowledge, which possesses a great breadth of general knowledge that most people should already know. ConceptNet contains millions of triplets in the form of (subject, predicate, object), where subject and object can be mapped to visual objects from our system by looking up their labels. Thus, we look for each object within a pair and extract the available relationship between them to refine the set of relationship proposals between the two objects. Let the pair ("person", "crosswalk") as an example in Figure , we can extract the node "cross the street".

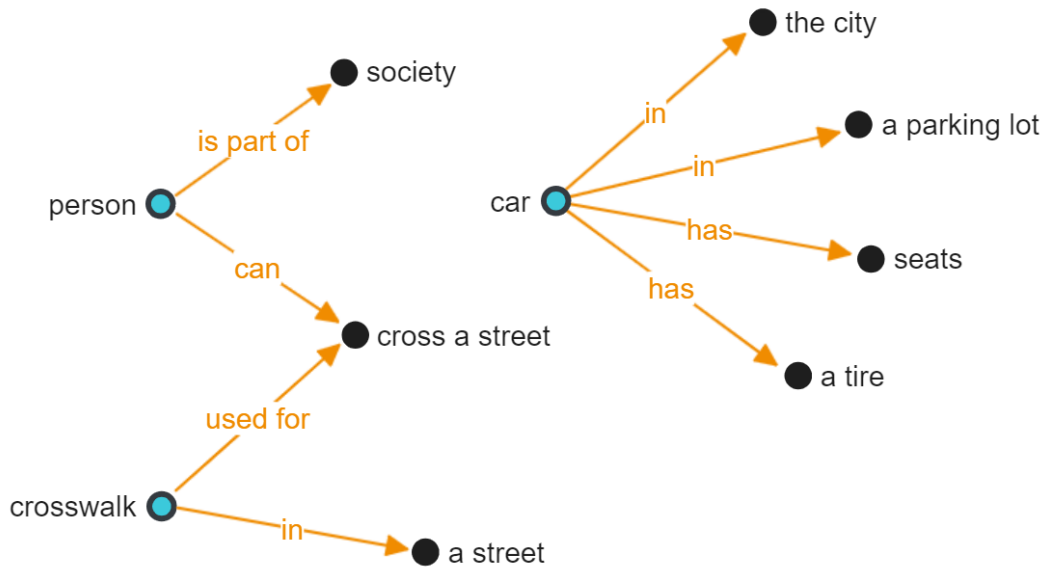


Figure 2-6. A subgraph of ConceptNet showing the connection between nodes

Relation Prediction

In the final stage, we perform a predicate classification to predict the relationship between each pair of objects. For this task, we first define a fixed set of relationships proposal, and then extend it using the common-sense knowledge from the previous section. Then, we apply a zero-shot predicate classification inspired by the work of Jiang and Taylor [Jiang23]. The focus of this work is to create an informative hierarchical structure that can divide object and relationship categories into disjoint super-categories in a systematic way. We divide predominant relationships in scene graphs into four super-categories instead of three as in the paper. That is, we collect the most frequent predicates and distribute them into "geometric", "possessive", "semantic", and the last one "common sense" which includes the predicates obtained from the ConceptNet. These categories can be explicitly utilized in the next steps. Table 3.4.2.1.1 shows an example of the predefined set of predicates. Moreover, Jiang and Taylor also introduce a Bayesian prediction head to jointly predict the super-category of relationships between a pair of object instances, as well as the detailed relationship within that super-category simultaneously, facilitating more informative predictions.

Table 3.4.2.1.1. An example of pre-defined set of predicates

GEOMETRIC	above, and, at, behind, between, covering, in, in front of, near, on, on back of, over, under
POSSESSIVE	attached to, belonging to, carrying, has, of, part of
SEMANTIC	across, against, along, covered in, eating, flying in, for, from, growing on, hanging from, holding, laying on, looking at, lying on, made of, mounted on, painted on, parked on, playing, riding, says, sitting on, standing on, to, using, walking in, walking on, watching, wearing, wears, with

Together with extracting the explicit features E and implicit features I of objects in the image, we also employ the MiDaS [Ranftl20] single-image depth estimation network to provide depth maps D for input images. The final image features $I' = \text{concat}\{E, I, D\}$ serve as the inputs to the

predicate classifier. Consider each pair of objects, the predicate classifier yields four predicates, one from each disjoint super-category, which maintains the exclusivity among predicates within the same super-category. These predicates will be ranked to select the most confident one as the predicted relationship for the current pair of objects. We also consider adding a coefficient, e.g. a value between 0 and 1, to control the dominance of the common-sense relation proposals. However, this step requires more carefully experiments to define the most optimize value for w .

2.4.2.2 Scene Understanding in Smart Factory

The objective of the smart factory use case is for autonomous mobile robots (AMRs) and other intelligent devices on the factory floor to detect typical objects in the manufacturing environment, understand their context in relation to the other objects around them, and thus operate more effectively in dynamically changing situations. By sharing knowledge of what they perceive with other intelligent robots and edge devices in the swarm, they can build a mutual understanding and context of their environment and better collaborate to achieve common goals, sharing scene information so they can model things they cannot perceive directly through their own sensors.

By understanding the scene in real-time the AMRs can navigate their environment, avoiding dynamically moving obstacles, and take appropriate behavior given a certain scenario. For example, if the AMR encounters an unexpected pallet blocking its way, it might inform the operation's staff of the obstacle and try to take another route. However, if the obstacle was a person, it might stop and issue an audio alert to request the person to clear the path. Most automated manufacturing environments are deterministic, that is products, material and machinery are placed at known defined locations. Having a deterministic environment simplifies manufacturing, but at the expense of flexibility and adaptability, two qualities that greatly aid smart factories. For example, it may not be efficient for a mobile product rack to be positioned exactly at the same position each time, therefore the AMR should be flexible enough to adjust to the current location; likewise, people may inadvertently change the position of products, which means the environment is no-longer deterministic, therefore the AMR should be adaptable enough to deal with the product where it is, not where it should be. Even with limited intelligence and scene understanding, the AMRs should be able to adapt to limited changes in their environment.

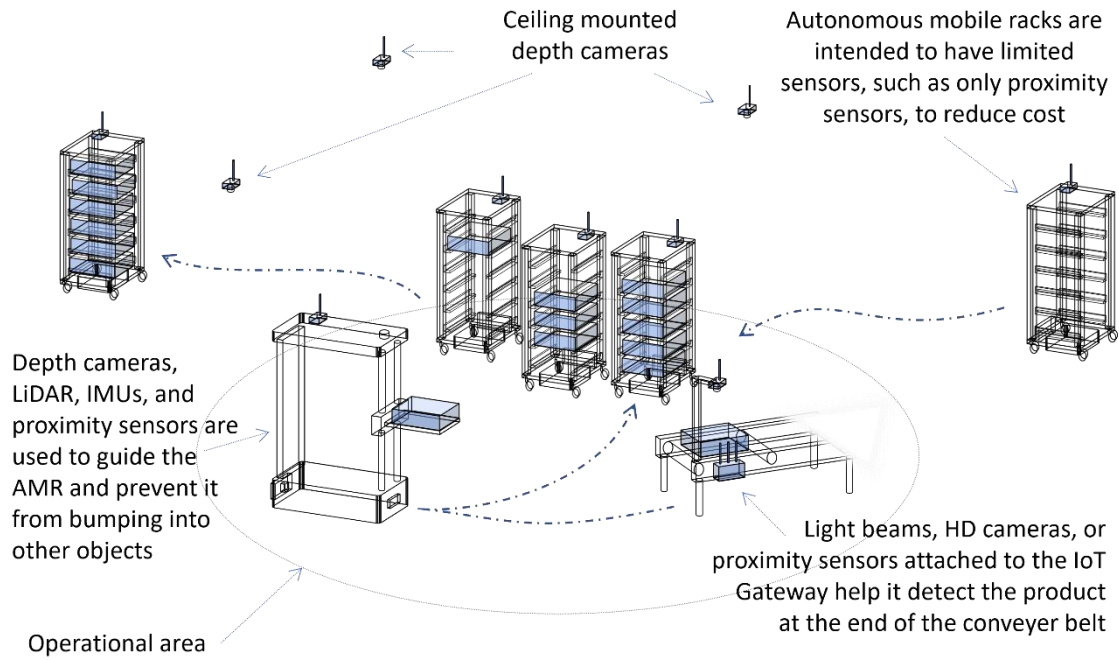


Figure 2-7. Sensors used in operational area

A wide variety of sensors may be used in the smart factory use case, as illustrated in Figure . They range from ultrasonics and pressure bumpers to detect proximity to objects, to high-definition optical cameras to detect product types, and 360° rotating LiDAR and IMUs for generating SLAM maps; but the two most important sensors are MEMS LiDAR cameras and stereoscopic depth cameras, as illustrated in Figure . Both types of sensors provide RGB images and depth maps (range to a target) in a single sensor. The LiDAR uses a scanning laser to detect range, whereas the stereoscopic camera uses stereo vision to triangulate key features in the image. They differ primarily in the greater range of the LiDAR camera; however, the range of stereoscopic cameras is improving year-on-year. Obtaining both types of data from the same sensor ensures that the two data streams have the same frame of reference, although they do have different resolutions.



Figure 2-8. MEMS LiDAR cameras (left) and stereoscopic depth cameras (right)

The proximity sensors are primarily used to detect objects at close range, and feed directly into the AMR’s emergency stop system. The high-definition camera images are used to detect serial and model numbers on the products, as well as other distinguishing features. These pieces of information are used to identify the specific product, or at least its model type. The 360° rotating

LiDAR and IMU is used to generate SLAM maps as the AMR explores and navigates its way around an unknown area, and most importantly, the AMR's location within it. The 360° rotating LiDAR also helps to detect potential obstacles at greater range than proximity sensors, however they can often be fooled by transparent surfaces, such as glass windows and doors.

Object detection and classification can be problematic, and identifying individual things, such as a specific AMR, can be even more challenging. To aid with object identification QR codes will be attached to some manufacturing equipment in prominent locations, as illustrated in Figure . The identification QR code is unique to a thing and is stored as a property of the thing in the Thing Description Directory (TDD). Therefore, if a QR code is associated with an object captured by the cameras, its properties can easily be added to the semantic graph representing the scene, greatly simplifying the identification process. Additionally, QR codes associated with calibration chessboards will be mounted to the factory floor at several locations. The QR codes are at known locations and poses are relative to the factory's origin and axis. A ceiling mounted camera will be able to see at least one floor mounted QR code in its field of view. Along with the chessboards these codes can be used to calibrate the cameras and determine the relative position of other objects in the scene. As with other objects in the scene, the floor mounted QR codes are stored as things in the TDD.

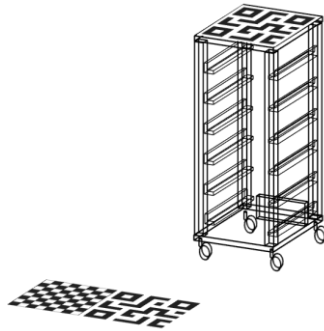


Figure 2-9. Rack with QR identification code and floor mounted QR code and calibration chessboard

The SmartEdge project will investigate a number of techniques to perform scene understanding, and the smart factory use case will demonstrate several of these techniques, including Semantic SLAM and 3D Semantic Modelling.

Semantic SLAM [Manh22] enhances the established robotic navigation method known as SLAM and is being developed by TUB, see section 5.4.1.2.2 for more details. This technique traditionally employs sensors like LiDAR to gauge distances to objects and surfaces, including walls and poles. The data, either as a 2D point line or 3D point cloud, helps a robot create an internal map of its surroundings. This map enables the robot to navigate without colliding with obstacles and to pinpoint its location, often utilizing landmarks and a Kalman filter. Although SLAM identifies objects and surfaces, it doesn't understand what they are. Integrating camera images with LiDAR data allows for object recognition. Identified objects are then linked with semantic information from the TDD, like collision domains and mobility status. By applying this process to various objects, the robot begins to attribute properties and behaviors, gaining a more sophisticated understanding of its environment, hence 'Semantic SLAM.' This approach is particularly beneficial in recognizing stationary objects, useful as fixed landmarks. Overall, Semantic SLAM provides the robot with a deeper insight into its environment, the objects within it, and their relationships.

3D Semantic Modelling is similar to Semantic SLAM in that it models a physical environment and understands the context of objects within it. However, unlike SLAM maps that are built up over time as the robot moves around the environment, 3D Semantic Modelling is built instantaneously using images and depth data from multiple sensor viewpoints at the same time, allowing the scene to be seen in the round. Each sensor viewpoint classifies the objects in the scene using object detectors and their surfaces using point clouds, which is represented as a semantic graph. Several semantic graphs from different positions are then fused together to infer a 3D semantic model of the environment, i.e., multiple surfaces are combined to construct the probable 3D shape of each object along with its classification. This type of model is particularly useful in dynamically changing environments, where the robot needs to modify its behavior in real-time in response to other independent actors, and where the robot possesses limited sensors or is unable to perceive specific areas of the environment. For example, a rack is inadvertently pushed into the planned path of the robot.

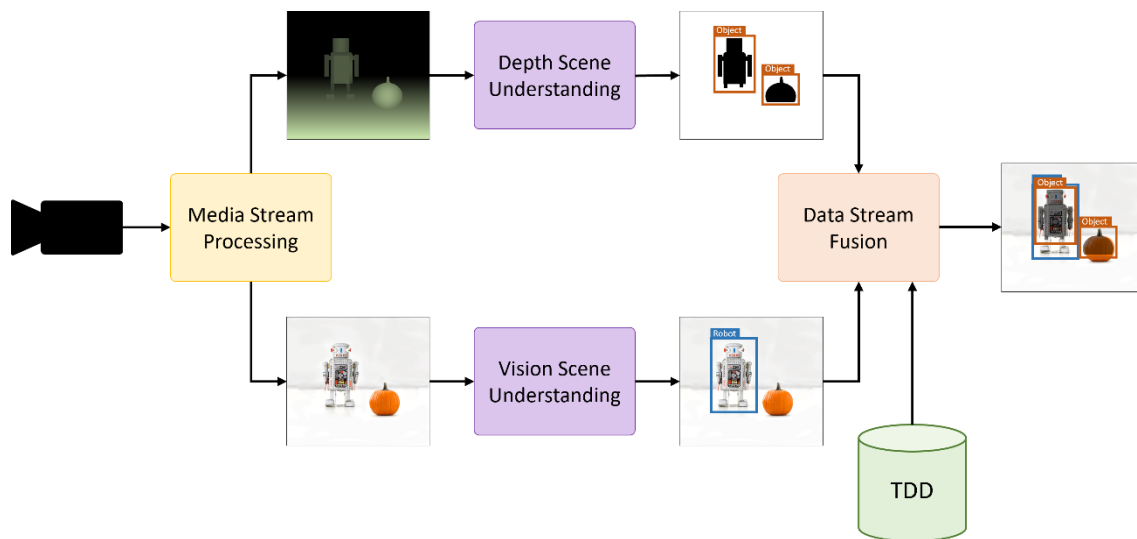


Figure 2-10. Semantic sensor stream fusion pipeline

Figure illustrates the semantic sensor stream fusion pipeline that will be used for the smart factory use case. Data streams from the image/depth sensors mounted on the ceiling will be pre-processed and split into their component parts, i.e., RGB and depth. The depth scene processing subtracts the background floor from the foreground objects, and so identifies probable objects and their surface. This process is aided by the fact that the factory floor is flat and uniform. Bounding boxes are added around each object identified. As with SLAM processing, the surface of objects is identified, but not classified. The vision scene processing uses object classifiers, such as Yolo, to classify probable objects in the scene. Again, bounding boxes are added around each classified object. Both scene understanding processes output semantic graphs in RDF format that represent some perception of the scene. Additionally, if the object detected is a QR code, it is passed to a QR code reader, and added to the vision semantic graph. The graphs are then fused using fuzzy logic, such as Intersection over Union (IoU) [Rezatofighi19], to associate objects to classifications. If objects were not classified, such as the pumpkin, they are given the class “unknown”.

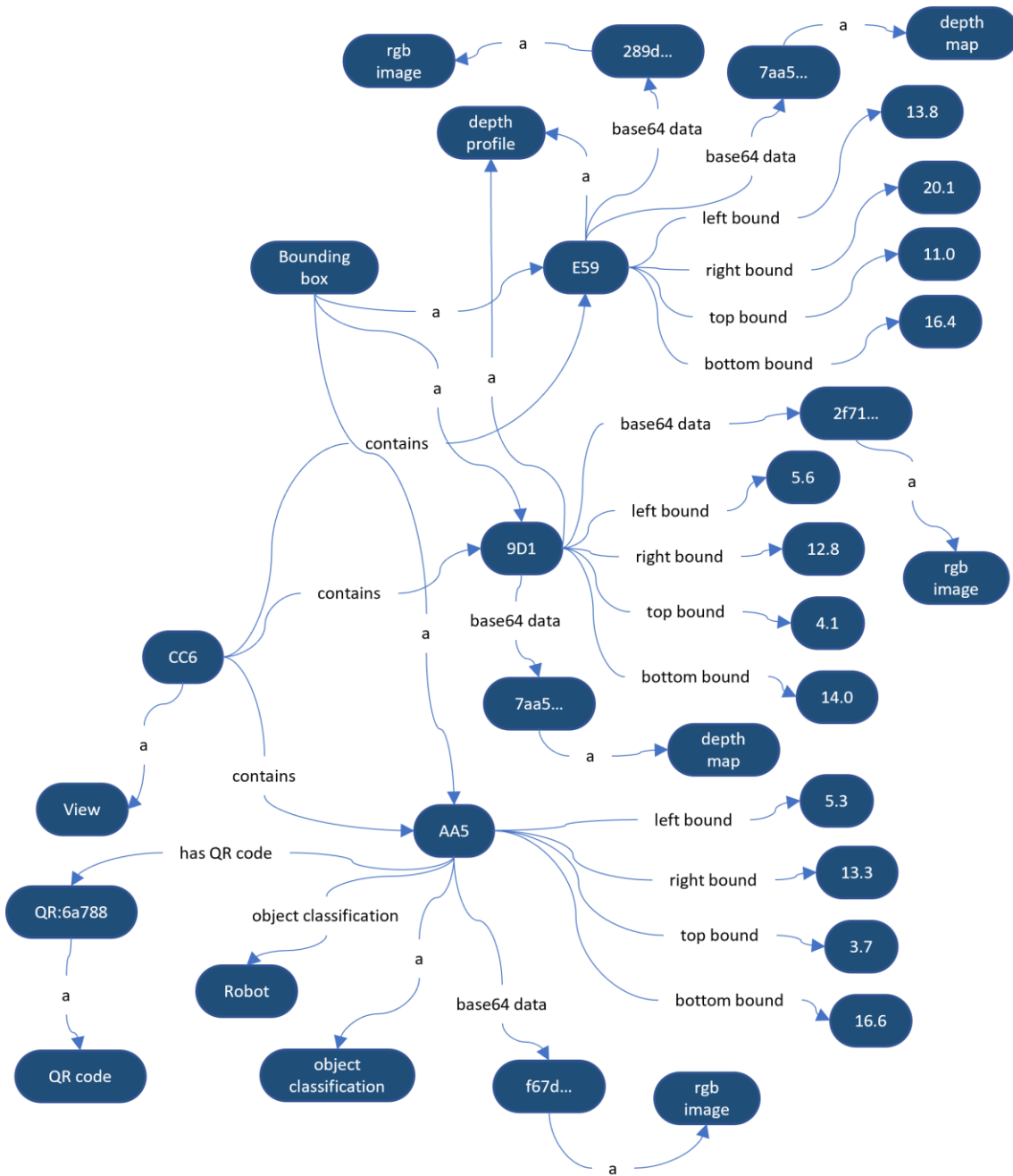


Figure 2-11. Semantic scene graph

The output of the semantic sensor stream fusion pipeline is a stream of semantic graphs, as illustrated in Figure . The graph can also be augmented with thing properties from the Thing Description Directory (TDD). The TDD is part of the Web of Things (WoF) standard [W3C24] supported by the W3C, which seeks to extend the concepts of Internet of Things (IoT) by providing a framework to define Things. SmartEdge defines Things as swarm nodes, independent agents, and objects that exist in the use case environment, and stores their properties, actions, and events in the TDD. For example, the weight of the robot and the behavior that it can move. The TDD properties enrich the semantic graphs and provide additional context for the detected objects. It is anticipated that the semantic stream processing pipeline can be performed in IoT gateways located close to the sensors. So, what is generated is a semantic stream of data describing the scene, and not the original raw images. This type of far

edge processing should help to reduce transmission bandwidth between swarm nodes, but this needs to be validated by experimentation. Additionally, sub-images from the bounding boxes may be linked to the semantic stream and transmitted with it. These sub-images may be useful for subsequent processing and require less bandwidth, as discussed in [Bowden22].

Semantic scene graphs from multiple ceiling mounted sensors/IoT gateways are streamed and combined in another swarm node with greater compute power. This node reconciles the semantic scene graphs from multiple image/depth cameras into a single 3D semantic model of the environment in a common frame of reference. This is a far more complex task as it uses projective geometry to place the surfaces of an object, as seen from each camera, in three-dimensional space. Based on the surface projections and the physical structural definitions of the things from the TDD, the process estimates the probable location and pose of the object and uses this to produce the 3D semantic model in RDF format. The physical structural definitions of the things in the TDD can be thought of as a digital twin of a classified object. This is currently an open area of research, but there are a number of promising techniques that will be investigated, such as Normal Aligned Radial Feature (NARF).

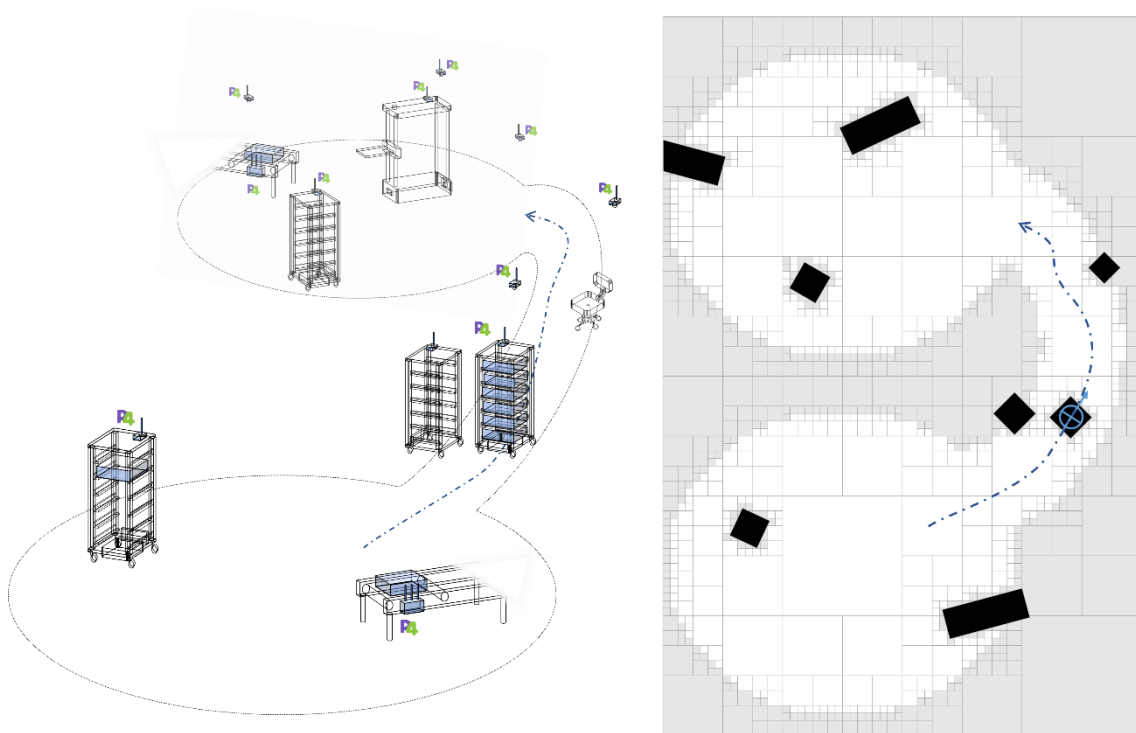


Figure 2-12. Illustration of rack moving between operational areas (left) and a 2D occupancy map (right)

Once the 3D semantic model of the environment has been generated, it can be shared with other nodes in the swarm, as they all have a common frame of reference, i.e., the same factory origin and axis, and a common semantic vocabulary. The model can be used for a number of different applications in the smart factory use case. For example, navigation of blind racks between operational areas. In this example a 2D plan projection (as seen from above) of the 3D semantic model is constructed to create a 2D occupancy grid, which can be used by the NAV2 component in ROS to navigate the autonomous mobile rack between operational areas, as illustrated in Figure . As the model is updated in real-time, it can also be used to avoid unexpected obstacles.

2.4.2.3 Integrate Multi-modal Sensor Data Sources

In line with UC2, various sensors both static and moving are deployed in Helsinki. This section outlines the requirements for developing an effective data fusion from these diverse data sources, especially for implementation of UC2 smart traffic. Below we explain the multimodal data inputs together with the expected output resulting from sensor fusion. Subsequently, UC2 will use and aggregate the sensor fusion outputs to calculate *traffic indicators* (queue length, vehicle density, option zone metrics, etc.). For this reason, the fusion will enable a comprehensive understanding of the environment, facilitating effective decision-making for traffic control.

It is important to note that each individual road-user might be detected by multiple sources at the same time, therefore the sensor fusion algorithm should consider this phenomenon. A typical example is a tram or a connected vehicle that is publishing its GPS geolocation wirelessly, also at the same time being detected by the road-side radar and camera.

A) Multimodal Inputs (coming from various sensing data sources):

Fixed Sensor sources (Fixed on roadside):

1. Radar:
 - Occasionally noisy, limitations in detecting pedestrians and bikes.
 - Detection frequency approximately 12 Hz.
2. Camera:
 - Weather-dependent noise but capable of detecting pedestrians and bikes.
 - Detection frequency varies (10 to 30 Hz).
3. Loop Detector:
 - No object classification, but low latency and high reliability.
 - Activates when an object passes over it

Moving Sensor sources (Mounted or Embedded in Vehicles):

1. Radar (distance to the front and back vehicle, narrow angle)
2. Camera.
3. Lidar (360 degrees, object detection).
4. GPS (reports geolocation)
 - a. GPS of V2X-enabled connected vehicles
 - b. GPS of Helsinki Region Transport (HSL) vehicles such as buses and trams
5. Inertial measurement unit (IMU) (can include acceleration, orientation, etc.)
6. Sensors connected to the CAN bus (status of tires traction, brakes, etc.)

B) Expected Outputs from the Fusion Process (Attributes of Interest):

The UC2 expects ideally the following outputs to be available from the sensor fusion process. Note that this is an ideal list and not all following values may turn out useful depending on the application.

For each Detected Moving Object (Road-users):

- Geolocation (latitude, longitude) of each unique vehicle.
- Speed
- Direction
- Acceleration

- Classification (Type) of road-users:
 - Vehicle type (Pedestrian, Bike, Motorbike, Car, Van, Truck)
 - Public Transport type (Tram, Bus, etc.): possible to match and fuse by combining (a) HSL data, and (b) Radar data.

For each Detected Static Object (Road Infrastructure):

- Geometry (e.g. zebra line's geometry).
- Geolocation
- Facing (angle)
- Classification (Type of object):
 - Left/right borders of the road
 - Stop line
 - Traffic signs (STOP, Speed limit, etc.)
 - Traffic light (Signals)
 - Crosswalk (pedestrian zebra line)
 - Lane separators

It should be noted that the vehicle on-board unit (OBU) can already fuse the following sensor data to some extent depending on its computational and software capabilities. The OBU integrates data from a set of vehicle-mounted sensors, including GPS and Accelerometer. The resulting information, encompassing location, speed, heading angle, etc., is transmitted via V2X communication to other vehicles and edge devices. For another set of vehicle sensors, consisting of Lidar and Camera inputs, the OBU performs object detection for both moving and static objects surrounding the vehicle. Regarding the surrounding moving objects, the OBU could estimate their location, speed, direction, and additional attributes. This information is useful for maintaining situational awareness. In the case of static objects, the OBU can perform object detection for various elements, including road geometry, lane geometry, barriers, road limits, traffic signs, and more.

We should also add here that the previous sections explain semantic scene understanding from cameras observing road-user traffic, which is relevant to obtaining a semantically valuable sensor fusion.

2.4.2.4 Media Stream Processing

The Media Stream Processing Pipeline in SmartEdge includes the software components required to make media streams from capturing devices for the different Use Cases like 2D Cameras, LiDAR Cameras, etc. from cars or robots available in the appropriate format for further components like Data Fusion. The following aspects need to be considered after media data are captured. Media refers mostly to video and image data:

- **Video Codec:** There are a variety of video codecs in the media industry with varying encoding/decoding performance and compression rate. H264 is one of the video codecs that is supported natively on nearly any device. H265 is a newer video codec which outperforms H264 in terms of compression rate, but it requires more processing resources for encoding and decoding. VP8, VP9 and AV1 are other free open codecs with different encoding/decoding performance and compression rate. Due to its wide support, it is recommended to use H264 as a common video codec to transmit the video data between capturing device and the edge.

- **Video Bitrate:** Encoded videos require different bitrates to achieve a certain quality required by the different use cases. With video codec and a category of content known, a smart encoding component predicts the required bitrate to achieve the required quality using a given video codec. The smart encoding component utilizes ML techniques to classify the captured content and predict the resolution and bitrate. Also, the capturing device's processing capabilities can be considered too.
- **Transmission Protocol:** There are different transmission protocols for media streaming relevant for different use cases depending on the requirements of the use case. HTTP based streaming formats like DASH (Dynamic Adaptive Bitrate over HTTP) or HLS (HTTP Live Streaming) are file-based (segment-based) media streaming protocols which are designed to server video content over a content delivery network (CDN) to a large audience with focus on scale. In optimal case, these protocols can achieve a latency of 3-5 seconds when used in combination with the Common Media Application Format (CMAF), which allows chunked-based streaming. SmartEdge Use Cases require real-time capability and therefore, the HTTP based protocols DASH and HLS are not suitable. The Real-Time Streaming Protocol RTSP is better suitable to support SmartEdge use cases. RTSP is supported natively by many CCTV and IP cameras but can be also installed as a software component on the SmartEdge client/device platform, that convert the encoded camera stream into RTSP format and transmit it over the network to the streaming server running on the edge of the SmartEdge network. RTSP can be configured to run on top of Transmission Control Protocol (TCP) and User Datagram Protocol (UDP) depending on reliability and latency requirements. In addition to RTSP, WebRTC is another protocol that can be used for real-time video transmission.

The Media Stream Processing Pipeline is provided in the diagram below (Figure).

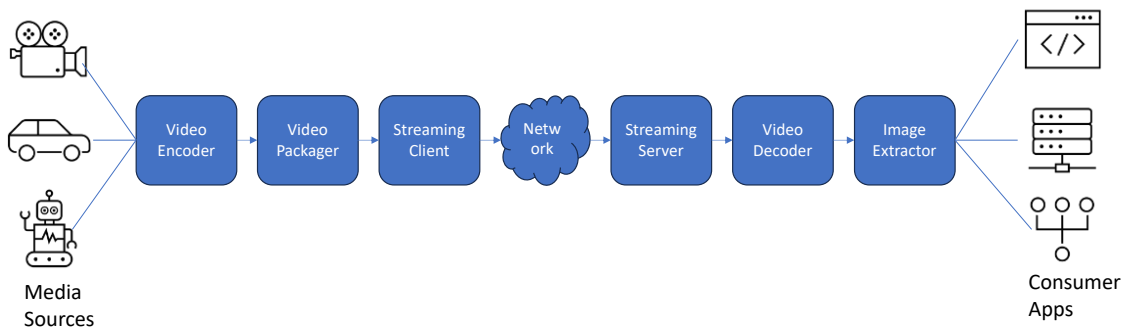


Figure 2-13. Media Stream Processing Pipeline

The main components of the Media Stream Processing pipeline are described below:

- **Video Encoder:** the video encoder receives raw video streams from capturing camera devices e.g. LiDAR cameras insight a car, built-in cameras in robots or even from standalone cameras. The video encoder then encodes the raw camera stream using a preconfigured video codec and provides a compressed video stream as output. In addition to video codec, resolution and bitrate need to be provided. The values of bitrate and resolution will impact the quality of the video and the latency as well. The

Smart Encode component may be applied also to calculate required bitrate and resolution depending on the quality of the capture content.

- **Video Packager:** the video packager is an optional component which is only required if Adaptive Bitrate Streaming (ABR) is required. The video from first step needs in this case to be encoded in different resolutions and bitrates for achieving different quality levels. The encoded streams will be then split into small segments that are packaged via state-of-the-art ABR formats like DASH or HLS. The length of the video segments has a great impact on the latency.
- **Streaming Client:** The streaming client is responsible for preparing the video streams for transmission over the network to the remote SmartEdge components for further processing. The streaming client and the streaming server (described below) need to use the same streaming protocol. In the case of DASH or HLS, the streaming client is just a simple HTTP client that posts the video segment to the streaming server. In the case of using other protocols like RTMP, RTSP or WebRTC, corresponding client and server libraries implementing these protocols needs to be used.
- **Streaming Server:** The Streaming Server is the component deployed in the SmartEdge System and can receive the media streams from client devices that act as media sources via the Steaming client component. The Streaming Client and Streaming Server must support a common transfer protocol like RTMP or RTSP and agree on using a common video codec. Many streaming client/server solutions already support multiple streaming protocols and video/audio codecs so that the applications can configure the most appropriate protocol and video/audio codec for the current use case. It is important to note that the Streaming Server component will serve streams from multiple source devices and make them available for the next component in the pipeline. Since media encoding/decoding/processing requires high processing resources, the media server may distribute the decoding task on multiple SmartEdge nodes where processing resources are available.
- **Video Decoder/Image Extractor:** the video decoder and image extractor component are responsible for decoding image frames from a video stream and provide them in an appropriate format to the next component in the pipeline. These components usually consume images either in raw format (bitmap) or compressed format like JPEG or PNG.

2.4.3 Semantic Data Stream Fusion

2.4.3.1 Adopting FAIR principles

In SmartEdge we adopted the concept of FAIR data, i.e. ensuring findability, accessibility, interoperability, and reusability of data.

To ensure the *findability* of data, use Uniform Resource Identifiers (URIs) to identify resources, including images and their associated metadata. These URIs provide unique and persistent identifiers for each resource, making it easy to find and access specific information, e.g. images or sensor devices.

We implement the *accessibility* of data and metadata by using standardized communication protocols and supporting the decoupling of metadata and data. Its publication practice makes it easier for the targeted user groups to access and reuse relevant data and metadata. The data, together with their metadata can be queried using standardized query languages, e.g. SPARQL and its extension to streaming processing.

To make our stream data *interoperable* across different datasets, we propose to use a unified ontology to capture different data source in different modalities. And at the same time, we maintain a mapping between this unified ontology with more specific ontologies in different use cases and/or components. We plan to reuse Wikidata as much as possible and add new terminologies, which are more specific to our project.

Since we utilize standardized data model and ontologies, the data can be *reusable*, firstly in different use cases, and potentially, in different projects.

2.4.3.2 Semantic Data Streams and Declarative Mapping Rules

The designed architecture for multimodal stream fusion, presented in this section, may be enhanced by leveraging Semantic Web technologies and the DataOps tool developed by WP3 in SmartEdge and described in deliverable D3.1. We identify and describe three main complementary aspects that support the designed solution for task 5.1 and can be leveraged in the implementation:

- i. semantic and declarative description of the data streams to be fused and accessed;
- ii. declarative transformation of the output generated by the Vision Scene Understanding component to the RDF output according to the target ontology;
- iii. enrichment of the graph extracted from multimodal stream fusion with additional information (e.g., contextual information) extracted from static data sources (e.g., datasets).

The first aspect refers to the semantic description through relevant ontologies of the data streams available as input for the multimodal stream fusion solution. Different ontologies are presented in the literature to this extent targeting the description of Web APIs and streams [VanAssche21]. In particular, the W3C Web of Things¹ recommendation can also be leveraged to describe Web APIs and streams implementing different protocols and security mechanisms. The recommended set of properties can be easily extended to support additional metadata (e.g., from the DCAT vocabulary²) to describe further the multimodal stream (e.g., encoding or location of the sensor/camera generating the stream).

The description of input streams using a common semantic model enables a declarative approach for their integration in the task 5.1 solution. The RDF description of the input stream can be indeed used to: (i) query the streams and find the suitable ones to be consumed for a specific task according to certain parameters, (ii) automatise the access and integration of data streams based on the information provided (e.g., connecting to an API by simply specifying the URL and the authorization mechanism to retrieve the data).

Figure shows an example RDF snippet describing a stream from a video camera using the W3C Web of Things specification, additional properties from other vocabularies and example properties that can support the implementation of the SmartEdge Schema designed in WP3. In this snippet, we describe the security mechanism for accessing the stream, the endpoint and the content type. Additional metadata about the location of the camera are provided using the Basic Geo (WGS84 lat/long) Vocabulary³. The SmartEdge properties describe the identifier of the node, the identifier of the swarm and the type of node. The RDF graph containing the description

¹ <https://www.w3.org/WoT>

² <https://www.w3.org/TR/vocab-dcat-3/>

³ <https://www.w3.org/2003/01/geo/>

of multiple multimodal streams can be queried to get information on specific streams, e.g., the ones belonging to a certain stream and/or located in a specific location.

```
@prefix td: <http://www.w3.org/ns/td#>.
@prefix hctl: <https://www.w3.org/2019/wot/hypermedia#>.
@prefix wotsec: <https://www.w3.org/2019/wot/security#>.
@prefix htv: <http://www.w3.org/2011/http#>.
@prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#>.

<#WoTCameraSecurity> a wotsec:APISecurityScheme;
  wotsec:in "header"; wotsec:name "apitoken".

<#WoTCameraSource> a td:PropertyAffordance;
  td:hasForm [
    hctl:hasTarget "http://example.com/camera-feed";
    hctl:forContentType "video/mjpeg";
    hctl:hasOperationType td:readproperty;
    htv:methodName "GET";
  ].

<#WoTCameraThing> a td:Thing ;
  td:hasSecurityConfiguration <#WoTCameraSecurity>;
  td:hasPropertyAffordance <#WoTCameraSource>;
  geo:lat "26.58";
  geo:long "297.83";
  smed:swarmNodeId "node-23dfsa-42";
  smed:swarmId "swarm-23dfsa";
  smed:nodeType smed:CameraNode .
```

Figure 2-14: Example of stream description using W3C Web of Things concepts and additional metadata.

The second aspect refers to the definition of declarative mapping rules to convert the outputs of the Vision Scene Understanding component to an interoperable RDF representation according to target ontologies. Indeed, the components responsible for analysing the input multimodal streams may generate a description of the current scene using heterogeneous and possibly not semantic formats. For example, a machine learning algorithm for object detection and classification may generate a JSON output classifying objects detected according to custom categories of objects. The conversion of these outputs according to a common semantic, i.e., their representation in RDF according to a target ontology, can be fulfilled using the naïve approach of a hardcoded script that processes the input to obtain the target output. However, this approach is not maintainable and, in the literature presented in D3.1, the solutions based on declarative mapping languages emerge as the best solution to address this need.

A pipeline, leveraging the DataOps tool implemented by WP3, can be implemented to support the declarative conversion of heterogeneous outputs to RDF using a common reference ontology. A set of declarative mapping rules is defined to configure the pipeline and the available components are responsible for their optimised execution. The declarative description is decoupled from its execution and thus more maintainable. Moreover, a DataOps pipeline can support the integration of converted data obtained from different data sources in a single RDF graph. A set of declarative mapping rules can also be used to explicitly define a DataOps pipeline that converts data between diverse RDF representations, e.g., converting RDF data using a different ontology to represent the same information.

Figure 2-15 provides an example of a JSON snippet representing the result of an object detection algorithm that identified a person and a car from a frame of a camera video stream. The conversion of such an input to RDF can leverage a DataOps pipeline to execute a set of declarative mapping rules, e.g., defined using the RDF Mapping Language⁴ (RML), and obtain a

⁴ <https://rml.io/specs/rml/>

corresponding RDF output leveraging a target ontology. In the figure, we leverage *schema.org* and *GeoSPARQL* vocabularies as an example and we refer to *Wikidata* entities for *Person* and *Car* to associate a IRI to labels identified by the object detection algorithm.

```

{
  "timestamp": "2024-01-29T12:30:45.678Z",
  "frame_number": 1234,
  "objects": [
    {
      "label": "person",
      "confidence": 0.92,
      "bounding_box": {
        "xmin": 120,
        "ymin": 80,
        "xmax": 200,
        "ymax": 300
      }
    },
    {
      "label": "car",
      "confidence": 0.85,
      "bounding_box": {
        "xmin": 300,
        "ymin": 150,
        "xmax": 450,
        "ymax": 220
      }
    },
    // ... additional objects
  ]
}

```

```

@prefix schema: <http://schema.org/> .
@prefix ex: <http://example.org/> .
@prefix geo: <http://www.opengis.net/ont/geosparql#> .

<https://www.smart-edge.eu/data/node-23dfsa-42/1234>
  schema:dateCreated "2024-01-29T12:30:45.678Z" ;
  ex:frameNumber 1234 ;
  schema:hasPart [
    ex:objectDetected <https://www.wikidata.org/wiki/Q215627> ;
    ex:confidence 0.92 ;
    ex:hasBoundingBox [
      a geo:Geometry ;
      geo:asWKT "POLYGON((120 80, 200 80, 200 300, 120 300, 120 80))"
    ]
  ] ;
  schema:hasPart [
    ex:objectDetected <https://www.wikidata.org/wiki/Q1420> ;
    ex:confidence 0.85 ;
    ex:hasBoundingBox [
      a geo:Geometry ;
      geo:asWKT "POLYGON((300 150, 450 150, 450 220, 300 220, 300 150))"
    ]
  ]
] .

```

Figure 2-15: Example of JSON input and RDF output for object detection.

The third and final aspect refers to the possibility of enriching the dynamic graph generated from the processing at runtime of multimodal streams with information extracted from static files, e.g., datasets. In this case, additional data sources are considered to improve the scene understanding, e.g., by providing contextual information that can help the algorithms in improving the accuracy. A DataOps pipeline can be defined also in this case to support the conversion of certain data according to declarative mapping rules and targeting the same reference ontologies adopted for the description of the scene.

This process can be implemented “online” by dynamically converting the needed information through a pipeline or “offline”. In this second case, the static data are converted and stored in an RDF graph (e.g., a triplestore) only once. A DataOps pipeline can be defined to query and retrieve at runtime the required data, e.g., a parametric REST API can be implemented using the DataOps tool and leveraged by the T5.1 solution during the processing of the streams.

Considering the example from Figure 2-15, additional information that can be used to enrich the RDF graph may refer to the description of other nodes in the swarm (e.g., geometric information), or to a list of objects that we know could be in the viewing angle of the camera and may be used to improve the accuracy of the result obtained from the object detection algorithm.

3 SWARM ELASTICITY VIA EDGE-CLOUD INTERPLAY

3.1 OVERVIEW OF EDGE-CLOUD INTERPLAY

As part of decoupling application logic from the underlying platform, Task T5.2 focuses on offloading part of the computations running in SmartEdge from edge nodes to potentially more powerful nodes (e.g., specialized SmartEdge nodes benefitting from some acceleration capabilities, or powerful nodes running in the Cloud) or better-connected nodes (e.g., central nodes or switches). Specifically, the task develops mechanisms to elastically use resources and offload specific stateful subqueries and AI operations from edge nodes to further nodes. To do so, T5.2 is based on three technical pillars: i) a dedicated declarative data exchange language and zero-copy networking protocol to exchange data with the best possible performance between nodes; ii) a set of accelerated operators to dynamically offload portions of the workload to specific accelerators or further nodes; and iii) a runtime to optimize some of the offloaded operations. Each of these three technical pillars is described in more detail below.

First, one has to ensure that the data transfer between nodes is as efficient (in terms of latency and CPU cycles) as possible, otherwise we run the risk of making the offload operation useless or even detrimental to the overall performance of the system (in case of high data offloading overheads). We leverage Remote Direct Memory Access (RDMA) in that context. RDMA is introduced in more detail below, but in a nutshell allows to transfer data from the memory of one node to another node in a high-throughput and low-latency way, by bypassing the operating system and cache hierarchy of the involved nodes. It enables the network adaptors of the nodes to transfer data from application memory directly to the wire and from the wire directly to application memory, eliminating intermediary copies in between (*zero-copy networking*). RDMA poses many problems in practice, however. In data intensive applications, it often results in fragmented and smaller data transfers that are detrimental to the overall performance [RLF+22]. To remedy this, the first part of T5.2. develops dedicated Declarative-RDMA (D-RDMA) protocols to speed up data transfer in SmartEdge; Instead of transmitting individual data fragments through RDMA (which wastes CPU cycles and memory bandwidth), the application specifies in a declarative, low-code language what data should be transmitted, letting the networking card optimize the transfer by issuing larger DMAs. D-RDMA takes advantage of the networking layer of SmartEdge and is implemented directly on top of it for a subset of nodes with powerful networking capabilities. We plan to develop a declarative language to enhance two classes of data transfers in the context of SmartEdge: transfers involving complex filtering operations, and transfers involving LiDAR point-cloud operations.

The second part of T5.2 looks into specific portions of the SmartEdge workload to offload. As implementing hardware-accelerated operators is both extremely time-consuming and intricate (because one needs to use low-level programming languages and optimize operations for a dedicated hardware architecture), we focus on a few use-cases representative of SmartEdge including AI and data-intensive operations. Specifically, we design accelerated operators for an intelligent road-side unit, as well as more generic accelerated solutions for graph and RDF operations.

The third and final part of T5.2 builds on the two points above to actually realize the offloading operation in SmartEdge. To do so, we build a runtime that uses D-RDMA to transfer data very efficiently between nodes, and then to optimize the runtime of the accelerated operations using various optimization strategies (e.g., physical, logical, or hardware-specific optimizations).

The rest of this section is organized as follows: we start by reviewing the KPIs and requirements related to this task below. We then present the relevant state of the art (in terms of modern data exchange protocols and heterogeneous computing) in Section 3.3, before introducing our detailed architecture and design in Section 3.4.

3.2 REQUIREMENTS AND KPIs

3.2.1 Requirements

This task is directly related to the following requirements:

SW-028 v1.1	To keep latency low the Cloud should not take on the role of swarm coordinator or orchestrator. The Cloud should implement the SmartEdge stack but may be restricted to only certain swarm protocols. There may be challenges if the Cloud has too much control over the swarm.			
	DELL	UC-3	High	4, 5

Specifically, the T5.2 will ensure that the Cloud only takes an acceleration role in performance-critical situations, while the management of operations (including the orchestration of the offload itself) will be handled by SmartEdge nodes directly.

SC-009 v1.1	End to end latency of controller status data should be less than 100 ms.			
	Aalto	UC-2, UC-3	High	4

Latency is key for several of our use-cases. T5.2. will make sure to keep latency low (below 100ms) for a set of specific mission-critical operations by leveraging advanced data transfer protocols and hardware acceleration.

SC-015 v1.1	A network protocol must exist to setup a data stream feed between devices in a swarm. This may be achieved through a publish and subscribe mechanism, or a streaming data protocol. For example, an AMR may wish to constantly receive video images from several ceiling cameras. There must also be a mechanism to stop the feed.			
	DELL, IMC	UC-3, UC-5	High	

The first technical pillar of T5.1 directly answers this requirement, by enabling very low-latency and high-bandwidth protocols between specific nodes.

CSI-013 v1.1	A swarm smart-node must have the ability to correlate sensor data from different sources on the same device in order to enhance the semantic understanding of the environment being observed, e.g. it should be possible to combine sensor streams from LiDAR, cameras, etc. in order to semantically annotate objects and other features in an environment in the smart-nodes internal knowledge graph, the LiDAR giving the physical location of the object or feature and the camera facilitating the classification based on the same frame of reference.			
	DELL	UC-3	High	3, 5
CSI-014 v1.1	The same requirement as CSI-013 by integrating sensor information derived from other nodes in the swarm.			
	DELL	UC-3	High	3, 5

The second technical pillar of this task investigates hardware acceleration for specific mission-critical operations. Specifically, we plan to develop toolchains to accelerate LiDAR data processing as well as graph and RDF processing in a generic way for several use-cases.

3.2.2 KPIs

ID	Descriptions	Contribution of T5.2
K4.1	Ability to free developers from specifying capabilities of hardware and sensors at the design phase of stream fusion pipelines with end-to-end latency guarantee (e.g., 20-75% lower latency to baselines [DTH+21, PET21])	Low-latency data exchange protocols and hardware acceleration will directly contribute to this KPI by enabling latency guarantees.
K4.2	Ability to elastically scale 200-500% better than the state of the art, e.g., [Cudre-Mauroux13, Duc21, Schneider22].	Hardware offloading will directly contribute to this KPI by enabling much better elasticity and scalability.
K4.3	Can dynamically optimize resource to be 50%-150% better in terms of computing resources and bandwidths;	Hardware offloading will directly contribute to this KPI by freeing CPUs and leveraging heterogeneous hardware instead.
K4.4	Lower the effort in building swarm intelligence with the target of reducing the coding effort in comparison to imperative programming paradigms by 80-90%, e.g., Python or C++, with SMARTEDGE low-code tool chain.	All components in this task will be based on declarative abstractions and hence will reduce coding efforts significantly

3.3 PRELIMINARIES AND STATE OF THE ART

Before delving into the technical specifications of our contribution, we give below a quick introduction to the technical foundations underpinning our work, in terms of i) data exchange protocols ii) declarative data exchange and iii) heterogeneous computing.

3.3.1 An introduction to modern data exchange and its challenges

The discipline of programming networks and exchange data used to involve simple concepts. Messages were exchanged through sockets, which invariably forced applications to use a sequence of simple send-receive patterns to communicate. Message forwarding was completely opaque to the application. Once a message was passed on to an initiating machine's *send* call, it would simply resurface on the desired target machine's *recv* call. How the network was structured between the initiating and target machines did not matter to the application.

Modern networks are very different. In the last decade, much has changed in networking to enable data-centric systems and applications at the cloud scale. Modern networks are larger, faster, more efficient, and offer (much) more services. Unsurprisingly, they bring many changes in how they are programmed.

The two technologies that best exemplify these fundamental abstraction shifts are *RDMA* and *programmable network devices*. The cloud industry has been using one or both of them for a few years because, quite simply, cloud providers cannot afford to forgo technologies that are efficient and deliver high performance. For edge computing, the story used to be different. In the past, it was difficult to access these technologies, and the learning curve was discouraging. Currently, these technologies are increasingly off-the-shelf, and a growing number of edge nodes are now supporting those new paradigms. The risk for both researchers and practitioners building innovative systems without these technologies is to find themselves behind systems that do.

RDMA stands for Remote Direct Memory Access, and, as the name implies, blurs the boundaries among servers allowing, for instance, for a process to read the memory of a remote machine. *Programmable network* devices allow applications to customize the way the network hardware behaves. They allow, for instance, semantics-based routing, e.g., routing a request to a server that is available rather than to a fixed destination address. Both technologies are key enablers in the context of this task and are introduced in more detail below.

3.3.1.1 An Introduction to RDMA

RDMA is gaining traction in data-intensive networks for all major cloud providers, enabling efficient routing of massive application traffic at scale with low latency. Simultaneously, the data-intensive market pivoted from on-premise to cloud-based solutions in recent years. Consequently, we believe it is an opportune moment for edge computing to adopt RDMA for constructing scalable and efficient systems.

RDMA allows a machine to directly access remote memory over the network without interrupting the CPU on the remote system. Specialized RDMA-capable network interface cards (NICs) are used to perform the memory accesses on the remote side. Conversely, the same technique is used on the sender to avoid unnecessary copies of data in the user-space into the

kernel-space. The sender essentially instructs the RNIC what memory needs to be sent, and the RNIC's DMA engine copies the data and places it on the wire. This offloading is called zero-copy transfer.

RDMA presents many advantages compared to traditional data transfer protocols. By bypassing the CPU, RDMA significantly reduces data transfer latencies. It also enables faster data transfer rates, which is beneficial for applications requiring high bandwidth such as SmartEdge. It reduces CPU overheads, freeing up CPU cycles for other tasks, which might be especially attractive in IoT and edge computing scenarios. Finally, RDMA supports zero-copy networking, where data is transferred directly from the send buffer to the receive buffer without intermediate copies.

RDMA can operate over various network protocols, including:

- **InfiniBand:** A high-speed, low-latency networking technology commonly used in high-performance computing (HPC) and enterprise data centers. InfiniBand is known for its high throughput and low latency and is a standalone network technology that requires specific adapters, switches, and cables, which might be doable in the context of SmartEdge for some of the use-cases (e.g., UC3 or UC5) only.
- **RDMA over Converged Ethernet (RoCE):** This protocol allows RDMA communications to run over Ethernet networks. RoCE maintains the low-latency and high-throughput characteristics of RDMA while leveraging the widespread infrastructure of Ethernet. There are two versions of RoCE: RoCE v1: Operates over standard Ethernet networks but requires a lossless Ethernet fabric, which can be challenging to implement in a large-scale or congested network. RoCE v2: Extends RoCE capabilities over Layer 3 networks, allowing for routing and larger-scale deployments. RoCE might be the preferred solution when using RDMA in SmartEdge between well-connected edge nodes and the cloud.
- **Internet Wide Area RDMA Protocol (iWARP):** iWARP enables RDMA over standard TCP/IP networks, allowing it to work with existing Ethernet and IP infrastructures without the need for a lossless network. iWARP is known for its ease of deployment in existing network architectures but may not achieve the same low latency levels as InfiniBand or RoCE. This might be the protocol of choice for high-throughput and low-latency data exchange between edge nodes in SmartEdge.

As is clear from the descriptions above, each of these protocols has its own characteristics and is suited for different network environments and requirements. In the context of SmartEdge, the ultimate choice between them will depend on a number of factors such as existing network infrastructure, the prospect of installing dedicated hardware on top of the current infrastructure, performance requirements, and costs.

RDMA verbs are the core operations or commands used in RDMA to initiate and manage data transfers between networked devices. They form the interface through which applications interact with RDMA-capable hardware. The most important RDMA verbs in our context are succinctly described below.

- **Send and Receive:** These are the most basic operations. The send operation transmits a message from a local memory buffer on one machine to a remote memory buffer on another machine, where it is received. The receive operation is posted in advance by the receiving side to specify the destination buffer for incoming messages.
- **RDMA Write:** This operation writes data directly from a local memory buffer on one machine to a remote memory buffer on another machine without involving the remote CPU. RDMA Write is typically used for one-sided communication, where the receiving side does not need to post a corresponding receive operation.
- **RDMA Read:** Similar to RDMA Write, RDMA Read allows a machine to read data directly from a remote memory buffer to a local memory buffer. Again, this operation is one-sided and does not require the remote side to post a corresponding operation.
- **Atomic Operations:** RDMA also supports atomic operations, which ensure that a read-modify-write sequence on a remote memory location is completed as a single, indivisible operation. Common atomic operations include Compare and Swap (CAS) and Fetch and Add (F&A). These operations are useful for implementing synchronization mechanisms, such as locks or counters, across machines.
- **Queue Pairs (QPs):** While not a verb per se, Queue Pairs are a fundamental concept in RDMA. A QP consists of a send queue and a receive queue. Verbs such as send, receive, RDMA read, and RDMA write are issued through these QPs. Each QP is uniquely identified and associated with a particular communication session between two endpoints.

The use of those constructs allows for the efficient, low-latency transfer of data between applications running on different nodes, making RDMA a key technology for applications requiring high bandwidth and low latency such as SmartEdge. Understanding RDMA verbs and concepts is technically complicated, however. In the context of low-code programming and SmartEdge, we believe that a higher-level approach – for example based on declarative programming constructs – should be preferred, which is precisely why we introduce Declarative RDMA (D-RMDA) in Section 3.3.2 below.

3.3.1.2 An Introduction to Programmable Network Devices

Programmable network devices represent a significant evolution in the field of networking, offering a level of flexibility and control that was previously difficult to achieve with traditional, fixed-function network hardware. These devices, which include switches, routers, and network interface cards (NICs), can be programmed to perform a wide range of functions and to adapt to new protocols or custom processing requirements without the need for hardware changes. Their programmability is crucial for supporting the dynamic, scalable, and efficient operation of modern distributed applications such as SmartEdge. In addition, new networked processing equipment (e.g., Programmable Network Processors and Data Processing Units) is emerging that allows for the customization of their operations in the network.

Many of those devices can be programmed in P4 (Programming Protocol-independent Packet Processors), a high-level language designed for programming packet processing operations. It

enables programmers to specify how devices process packets, including defining custom packet headers and specifying the processing logic for packets as they traverse the network device.

The programmes that these new devices can implement are, however, limited. First, NICs and switches are not general-purpose computers. Most adopt peculiar computing models, which may or may not be able to express the computations worth unloading (for instance, most loops and control flows are severely limited in P4, due to the feed forward and real-time constraints of networking hardware). Second, this type of equipment has tremendous I/O power but is limited regarding memory size and the length of programs they can support. Even with these restrictions, several system areas can benefit.

Our focus in the context of this project will be on networked data processing, analytical workloads, and Machine Learning workloads. For instance, assume an application where a swarm of edge nodes process analytical queries in parallel. In terms of data processing, a modern, programmable switch can be used to offload complex analytics operations on behalf of the edge nodes through dedicated algorithms [Lerner2019]. By using the switch, we swap what would have been a sophisticated distributed computation (e.g., complex aggregates on the nodes) with a central, simpler one (aggregation on the switch). We say, in such cases, that the network is accelerating the workloads.

Machine Learning has gained enormous importance and can, unsurprisingly, benefit from network programmability as well. For instance, parameter aggregation plays a crucial role in the training phases of Machine Learning workloads. During such time, the servers must communicate to collectively update the parameters (coefficients) they are calculating in parallel. The communication pattern in question would normally be all-to-all message exchanges, whose quadratic factor creates severe scalability problems. This aggregation can be implemented to use the network instead, implementing these so-called parameter servers for ML training on a switch. We will discuss further ML acceleration use-cases below that can be leveraged in the context of this project.

3.3.2 Declarative Data Exchange

As pointed out above, the DMA part of RDMA stands for Direct Memory Access. It refers to the ability of a network card (among other devices) to read and write data from a host's memory without CPU assistance. RDMA's performance depends on efficient DMAs in the initiating and target hosts. In turn, a DMA's cost is almost always proportional to the length of the data transfer. The exception is small DMAs, which suffer from high overheads.

However, many data-intensive operations often generate small DMA operations when using RDMA canonically. The reason is that the data they transmit is seldom contiguous by the time transmissions occur. Modern data-intensive systems avoid this problem by copying data into large transmission buffers and issuing RDMA's over these buffers instead. Doing this requires a substantial amount of CPU cycles and memory bandwidth, which might not be possible on many SmartEdge scenarios. To solve this issue and to provide a low code, effective data exchange framework to SmartEdge, we extend below D-RDMA, a declarative version of RDMA that FRIB recently proposed [Ryser22]. D-RDMA is declarative in that it specifies what data to transmit but not the DMA schedule to do so. The approach leverages some accelerator (e.g., a smart NIC) to group data fragments into larger DMAs and produce the same packet stream as regular RDMA.

The problem we tackle through D-RDMA can be summarized as follows. RDMA can be very CPU intensive if the data to transmit is scattered in many non-adjacent, small chunks. In that case, the system has to point to each chunk by filling in some RDMA control data structures: for each message, it has to create a Work Request (WR), and for each data chunk within that message, a Scatter-Gather Element (SGE). If the chunks are relatively small, filling these control structures, and transferring the small chunks which they refer to, can dominate the cost of an RDMA. Figure Figure (left) depicts this scenario.

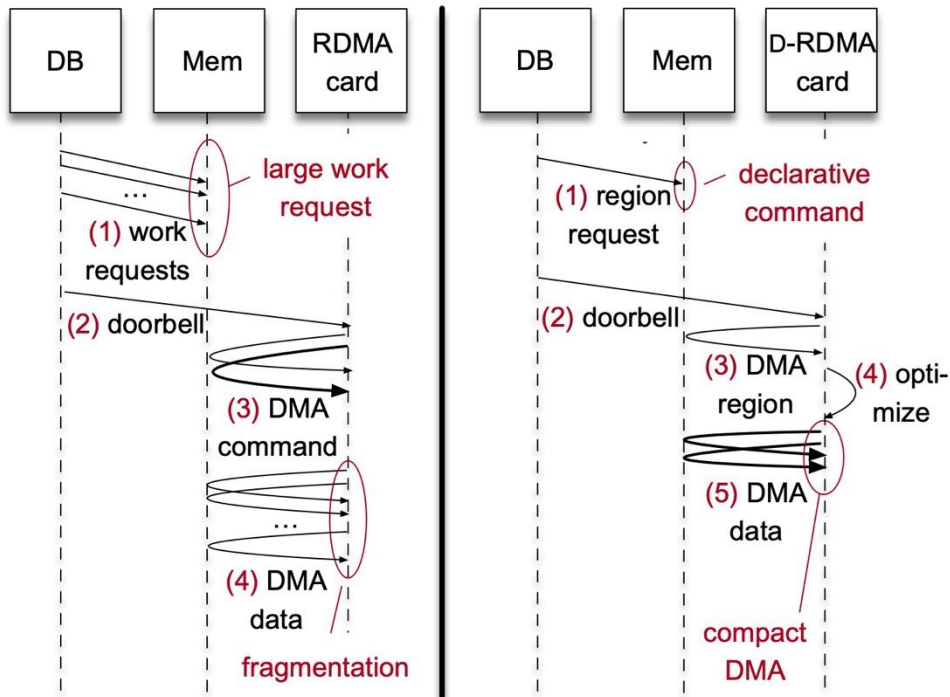


Figure 3-1: (left) standard RDMA forces the database to enqueue work requests for every fragment to be transmitted (1). Once enqueued, the card gets notified that new commands are waiting through a mechanism called a doorbell (2). The card proceeds by pulling the large commands from the submission queue (3). It then transfers one fragment at a time, regardless of optimization opportunities (4). (right): Declarative RDMA (D-RDMA) compactly declares in a low-code fashion the regions to transmit (1) and notifies the card the same way (2). The card pulls much shorter commands from the queue (3). In turn, the card looks for fragment coalescing opportunities and performs much larger DMAs should the opportunity arises (4, 5). The packets produced by the two approaches are identical.

Unfortunately, fragmentation, high-overheads and high-complexity are common issue when using RDMA in data-intensive systems. To remedy this, we proposed an extension of RDMA, called D-RDMA (for Declarative-RDMA) to both simplify and optimize the use of RDMA. The original D-RDMA approach extends RDMA with control structures that point to larger regions than a single message at a time and that contain both data and gaps. We call these structures Non-Contiguous Regions (NCRs). The D-RDMA extensions encompasses (a) these new structures, which are compatible with the verbs API, (b) new verbs that were missing from standard RDMA, and (c) an initial runtime to support them.

The simplest and most important NCRs are called *Strided Regions*. The rationale for this is the following: RDMA's work requests can be seen as a rudimentary language that can only describe contiguous regions. To make it more expressive, Strided Regions are a construct to capture whole regions that present data and gaps in regular patterns. Strides Regions in particular, and NCRs in general, are represented by data structures that replace the combination of WR and

SGEs with richer data descriptions. In other words, they are intended to be used instead of WR on certain RDMA verbs to make RDMA both more declarative and more efficient. A Strided Region can be defined using a base pointer, a period made of one or more elements, the width of the elements, and a stride. The stride is described by a frequency, e.g., 1 every 2 elements, and an optional start position, if different than the base address. Strided Regions are expressive enough to handle many data transfers in SmartEdge and other data-intensive projects. Figure gives a simple example of such a strided region.

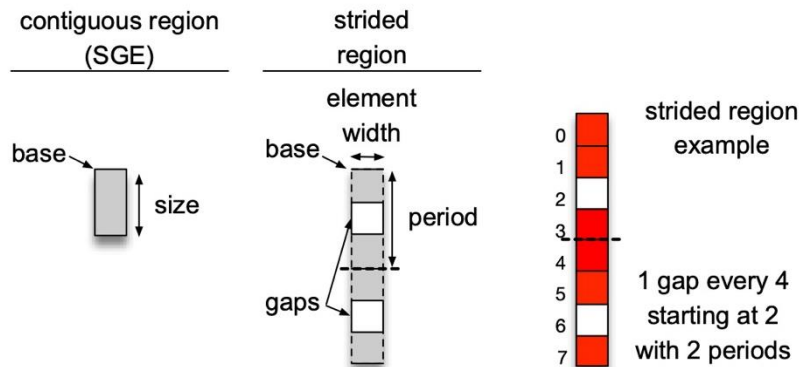


Figure 3-2: Contiguous Regions are insufficient to capture complex data patterns. Non-Contiguous Regions, such as Strided Regions, can be used to describe data and gaps in a compact way, declarative, and high-level manner and to optimize RDMA.

A D-RDMA request containing NCRs is handled by a specific runtime on a node. Figure 3-3(a) gives an overview of the workflow from a system's point of view. First, the application sets up the connections (queue pairs) to the remote hosts as it would in an RDMA scenario. It can then use the verbs API to send transmission instructions to the card. Certain verbs would take Non-Contiguous Regions (NCRs) to describe the requests. Upon receiving an NCR-based request, the runtime in the node forwards it to an optimizer, which determines the fastest DMA schedule to bring the data from the host. The runtime then executes this DMA schedule, and, as data arrives, it assembles the payloads contained in the NCRs before forming and sending out the packets.

Internally, the runtime comprises five components, shown in Figure 3-3 (b). Two of these components are similar to those we would encounter in a regular RDMA setting: the DMA Engine is responsible for transferring data from the host's memory into the network card's; and the Packetizer envelopes payload data with headers and trailers for the network protocol the card is handling. The third component, the Segmented Memory, is also present in regular settings but it is implemented slightly differently in D-RDMA: it considers smaller, independent memory buffers, which are used by the DMA engine to write data in a striped way. The remaining two components, the Optimizer and the (Payload) Assembler, are extensions required to process D-RDMA and are respectively responsible for deciding whether to transmit data chunks individually or to regroup them, and to dynamically assemble the payload of each packet to be transmitted.

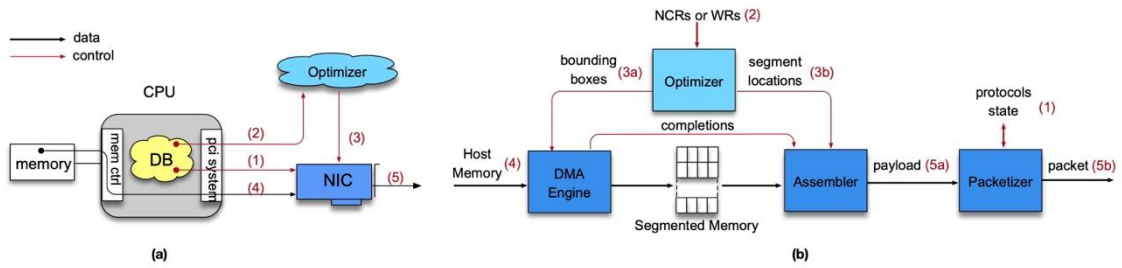


Figure 3-3. The life of an operation in the D-RDMA runtime from a system's perspective (a) and from a NIC's perspective (b). The application sets up a connection as usual (1). It uses declarative, Non-Contiguous Regions instead of SGEs to post work to the card (2). The card determines a DMA schedule upon receiving the NCR list (3,3a,3b). The card issues the DMAs (4). The card uses the row window for that request to find and packetize the data (5,5a,5b). We note that the optimizer can be implemented in software (e.g., at the driver level) or in hardware.

As a result, D-RDMA can describe data transfer declaratively, and can operate the transfer much closer to line speed, even for fragmented data and data-intensive scenarios such as those presented in SmartEdge. We will revisit D-RDMA in Section 3.4.4 below, where we describe specific D-RDMA extensions we have designed to handle advanced SmartEdge operations.

3.3.3 Heterogeneous, Low-Code Computing

In the end, the underlying vision behind task T5.2. is to combine recent advances in networking and hardware platforms with high-level, declarative, and low-code constructs. We call this vision *heterogeneous, low-code computing*. The idea is to leverage a wide range of heterogeneous computational platforms to run distributed workloads as efficiently as possible, leveraging CPUs on various nodes but also programmable networking devices and hardware accelerators. To do so, we systematically make use of declarative constructs and high-level programming to make such workloads easier to deploy and to invoke. We describe the overall architecture we have designed to make this vision a reality next in Section 3.4.

3.4 ARCHITECTURE AND DESIGN

3.4.1 General architecture

The general architecture for the work related to this task is described below. Mirroring the three technical pillars introduced above, our architecture is decomposed into three main components:

1. Advanced networking components extending the general networking layer of SmartEdge by leveraging next-generation data transfer powered by D-RDMA to declaratively invoke high-performance data transfers between nodes
2. Accelerated operators – taking specific portions of the SmartEdge workload and accelerating them in the cloud or using dedicated software accelerators

3. A runtime to; the role of the runtime will be to identify offloading opportunities based on a declarative specification in the recipe of the functionality, and then to optimize the offloading through a compilation process.

3.4.2 Integration with SmartEdge generic architecture

All three components directly fit in the general SmartEdge architecture:

1. The D-RDMA component runs on SmartEdge nodes that have RDMA capabilities
2. Accelerated operators run on specific nodes that either have higher P4 capabilities or possess dedicated hardware acceleration capabilities (such as FPGAs)
3. The optimizer can run either on a specific node boasting superior connectivity and P4 capabilities, or directly on the orchestrator itself depending on its connectivity and capabilities.

All components have been designed with low-code capabilities and they all feature higher-level, declarative interfaces:

1. D-RDMA is by definition declarative; D-RDMA operations can be directly serialized in a recipe and invoked in a declarative manner by a SmartEdge node
2. Accelerated operators will all be designed using declarative paradigms also (see examples below in Section 3.4.6)
3. The optimizer will also be designed using a declarative paradigm; it will receive as input a complex set of operations (e.g., a complex declarative query), which it will then optimize and run using, again, declarative specifications on some accelerator.

3.4.3 Integration with Use-Cases

3.4.3.1 Integration with UC2

Our first focus for this task is for UC2, though we plan to design solutions that will be as generic as possible and that will be useful for most use-cases. FRIB visited AALTO and CONVEQS in Summer 2023 to understand both the specificities of the use-case and the hardware acceleration that would be possible in that context.



Figure 3-4: Some of the nodes and hardware devices available in Helsinki for UC2 on the road (a) and in instrumented cars (b).

Both the equipment on the road (see Figure 3-4 a, which depicts road units deployed on road pillars) and equipment in instrumented cars (see Figure 3-4 b) support multi-modal sensing (including cameras, LiDARS, and GPS sensors) and could include additional equipment for hardware acceleration (e.g., in the form of low-powered Xilinx FPGAs or P4 accelerators).

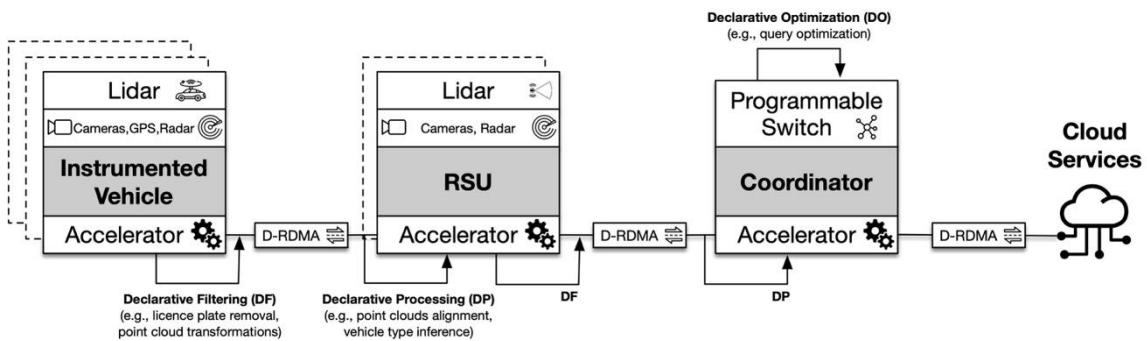


Figure 3-5: Integration of T5.2 with UC2; offloading will be powered by D-RDMA between edge nodes and further edge nodes with specific accelerators; Cloud nodes could also be used in that context.

Figure 3-5 gives an overview of how T5.2 integrates with UC2. Offloading data will leverage D-RDMA. In addition to fast data transfers, D-RDMA will be extended with filtering capabilities, in order to efficiently identify and remove portions of the raw data that can be cut out prior to transmission (called Declarative Filtering [DF] in Figure 3-5). In the context of UC2, license plate numbers could for example be filtered out prior to transmission (for privacy reasons), or part of a point cloud coming from a LiDAR installed on an instrumented car. See Section 3.4.4 for technical details on Declarative Filtering. More powerful nodes, such as an intelligent Road Side

Unit (RSU) can then take over the computations using hardware acceleration (called *Declarative Processing [DP]* in Figure 3-5). Advanced operations, such as point cloud alignment or vehicle type inference could then be run on a hardware accelerator on the RSU. A dynamic optimizer, running on a Coordinator or some powerful node in the swarm, optimizes the execution plans of offloaded operations whenever possible. In addition, Cloud nodes could be used for some of those operations whenever the swarm does not possess enough hardware resources to run the task.

Finally, Oxford plans to contribute to this task by offloading the construction of an integrated data structure (view) gathering information from several smart devices as Figure 3-6 depicts.

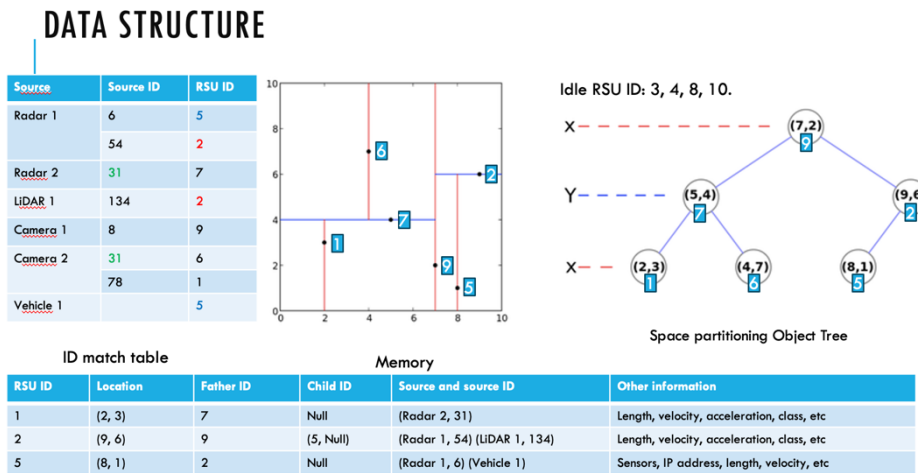


Figure 3-6: An integrated data view of various smart components for UC2

3.4.3.2 Integration with UC3

Use Case 3 (Collaborative Robotic Moves) aims to offer autonomous robots controlled with a swarm intelligence system able to grant superior reliability, efficiency and security in a smart factory scenario. Robots, in this scenario, aim to be autonomous in decision-making procedures and will be able to interact with other robots and with human operators.

Human-device interaction must occur granting full respect of an individual’s privacy on top of any other data analysis and exchange performed by the SmartEdge service. Image processing necessary to grant privacy should not impair communication rate within the swarm intelligence system.

To offer respect of privacy, any acquired image and/or video must be processed with proper face-detection and blur algorithm before any further analysis or communication performed by the swarm.

CNIT plans to contribute to this task by implementing a face blur detection and blur algorithm using OpenCV-compatible libraries, aiming to make it run on the ARM processor core available on the BlueField-2 DPU. The algorithm uses a YuNet detection model to perform the face recognition task (see Section 3.4.6.1 for details).

3.4.4 D-RDMA extensions

As introduced in Section 3.3.1.1, RDMA and its declarative extension D-RDMA, are state-of-the-art solutions to offload data to further nodes in SmartEdge. To accelerate the use-cases, we designed two important extensions of D-RDMA:

1. **Declarative filtering:** the idea is to extend the relatively simple capabilities of D-RDMA, namely Strided Regions (see above), to much more powerful online filtering operations. In this context, we plan to extend the filtering capabilities of D-RDMA to multiple dimensions. As a consequence, application developers will be able to declare what data they want to transmit through RDMA, including advanced constructs to filter out some of the data (e.g., by filtering out sensitive plate numbers on images for UC2, see Figure 3-7) on-the-fly while the transmission occurs. This functionality presents three key advantages: i) it is declarative and easy to use ii) it can filter out sensitive information on the sensing nodes directly without propagating this information any further and iii) it is extremely efficient as the filtering out, and then the exchange of data will be accelerated.

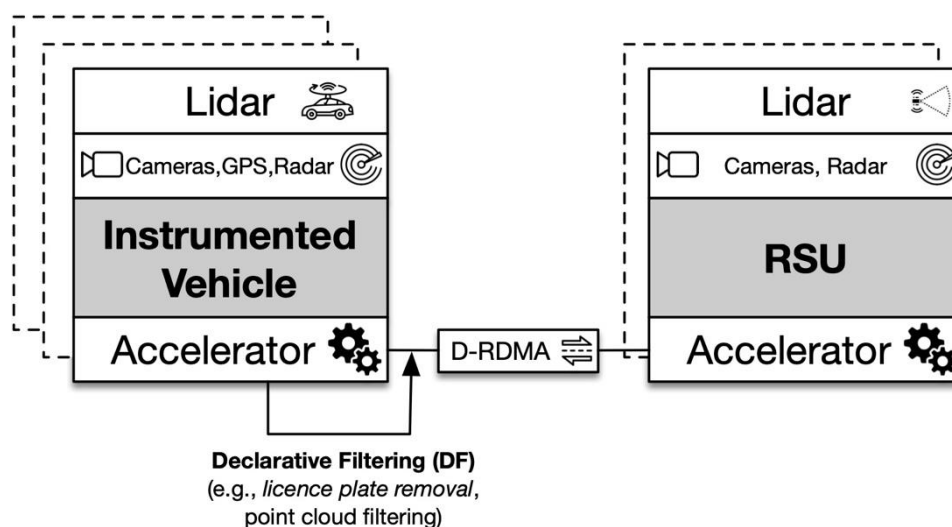


Figure 3-7: An example of a D-RDMA extension to accelerate filtering and exchange of data for UC2

2. **Point Cloud operations:** LiDARs are powerful and increasingly used edge devices that generate loads or data (e.g., 600,000 points per second on a simple LiDAR in Single Return Mode) in real-time. Raw LiDAR results are often called *point clouds*, i.e., sets of data points in a 3D coordinate system. Point clouds provide a wealth of information, that is however difficult to handle at the edge due to its size, velocity, and complexity. In this context, we plan to extend D-RDMA with point cloud primitives also. We plan to support to key functionalities: i) filtering point-clouds through D-RDMA focusing on regions of interest (e.g., some angle or region that is of particular interest for analyzing

car traffic at a junction for UC2) and ii) providing a basis for more advanced AI applications by accelerating Kalman filters through D-RDMA and an FPGA.

3.4.5 CXL Extensions

The last piece of technology that we plan to leverage to accelerate the offloading operation in the context of this task is CXL. Compute Express Link (CXL) is an open standard for high-speed, high-capacity CPU-to-device and CPU-to-memory connections, designed for high performance across components. CXL is built on the serial PCI Express (PCIe) physical interface and includes PCIe-based block input/output protocol (CXL.io) and new cache-coherent protocols for accessing system memory (CXL.cache) and device memory (CXL.mem). CXL allows systems to maintain coherence between a host's memory and memory from attached peripherals. Our use of CXL in the context of SmartEdge will be more advanced though.

We designed an extension of CXL called *CXL kernels* [Lee24] to automate part of the offloading operations in SmartEdge. The idea is to leverage advanced hardware operation to streamline read/writes at the application layer. Nodes will be able to expose a Database Kernel (DBK) such that read/writes against some specific memory range would trigger data-intensive computations that the kernel would perform directly inside the device. The nodes will use coherence traffic to monitor requests, prepare ahead of time, and ultimately answer data requests more efficiently. We believe that CXL and Database Kernels can support a new generation of heterogeneous data platforms with unprecedented efficiency, performance, and functionality.

3.4.6 Offloading computations

Once D-RDMA has been activated to offload data to a further node, the remote node can start taking over some computation. The whole idea behind Task 5.2 is to offload computation to special nodes that are in some sense more powerful or better connected. In the context of this project, we plan to focus on wide array of offloading operations. Beyond specific offloading operations like those occurring on the Intelligent Road Side Unit leveraging FPGAs (see Section 5.3.3. in Deliverable D4.1), or face blurring operations on a DPU (see below), we plan to focus as much as possible on generic offloading operators that can be useful for various use-cases. Since we know that many devices in SmartEdge will have some P4 capabilities (see D2.2), we focus as much as possible on advanced and generic P4 functionalities in the following.

3.4.6.1 Offloading Face Blurring on a DPU

As introduced above in Section 3.4.3.2, CNIT contributes to this task by implementing a face blur detection and blur algorithm using OpenCV-compatible libraries, aiming to make it run on the ARM processor core available on the BlueField-2 DPU. The algorithm is based on

face_detect5 (https://github.com/opencv/opencv/blob/4.x/samples/dnn/face_detect.cpp) and it uses a YuNet detection model to perform the face recognition task.

The YuNet detection model leverages on an open-source library for CNN-based face detection in images. The CNN model has been converted to static variables in C source files. The source code does not depend on any other libraries, and it may be compiled on any platform with C++ compiler.

The model is designed to be light-weight, fast and accurate, granting an accuracy between 0.77 and 0.89. It allows to recognize face of pixel between around 10x10 to 300x300 pixels due to the training scheme and can perform on multiple faces in the same picture. Performances of the CNN-based face detection method on intel CPU are reported online (<https://github.com/ShiqiYu/libfacedetection>).

Preliminary performance results of the face blur algorithm have been tested using the perf tool available on Linux, and results are reported below:

Performances for face_blur algorithm tested on an image		
FSP	25.7	
Time elapsed	0.0675 s	
	#	
Task-clock	364,47	5.40% CPU utilized
Context-switches	373	1.15 /sec
CPU-migrations	19	45.35 /sec
Page-faults	19210	46.09 /sec

Data in the table are median results of five consecutive tests. After detecting faces, the algorithm blurs the detected faces, granting individual's privacy before any other kind of image processing. A sample image is reported below (see Figure 3-8) to provide an example of the algorithm process:



Figure 3-8: Sample results for the offloaded face blurring operation showing the original image (left), the image with faces detected by the algorithm (center), and the resulting blurred image (right).

The algorithm has been tested on images and videos, and now is intended to be tested in a communication scenario, powering the process and data exchange with RDMA-based communication. Testing performances in terms of communication rate and data loss as well as

⁵ https://github.com/opencv/opencv/blob/4.x/samples/dnn/face_detect.cpp

memory consumption will represent an important step before the application on the swarm intelligence system.

3.4.6.2 Offloading Data-Intensive Operators to Accelerators using P4

In data-intensive systems, operations take the form of trees of operators. Each of those operators take as input a series of tuples, applies some operation (such as a selection, projections, join, or advanced operator like some AI processing) on this input, and returns transformed tuples as output. These operators can be parallelized on several (potentially many) machines in networked or cloud setups, or can be accelerate using specific nodes.

The main accelerators we will look into in the following are programmable switches supporting P4 operations (but note that the techniques we design below can take advantage of further P4 accelerators like DPUs or smart NICs). A programmable hardware switch is a platform unlike any other. It is divided into two semi-independent units, a control plane and a data plane, as Figure 3-9 depicts. The control plane is responsible for management tasks, e.g., bringing switch ports up or down. It usually consists of an x86 machine, an Intel Xeon in most cases, and it can run a common Linux distribution. The control plane functionality is available through C and Python libraries provided by the switch manufacturer.

The data plane is the component that receives packets from the network ports and forwards them back to their destination ports. The forwarding decision is the result of a computation— a networking protocol. In a programmable switch, the networking protocols are expressed as programs. These switches come with SDKs that can compile such programs into binaries they can run.

To explain how to program the data plane, we need to introduce a few concepts. The reason is that the programming model the switch supports is quite unique. The data plane consists of shared-nothing units called Match-Action Units (MAUs or, interchangeably, stages) arranged in a pipeline. An MAU is for a switch what a core is for a general-purpose x86 CPU. MAUs, however, have many constraints. Chief among them is that they can only send their results to the next MAU in the pipeline and can only receive input from the previous one.

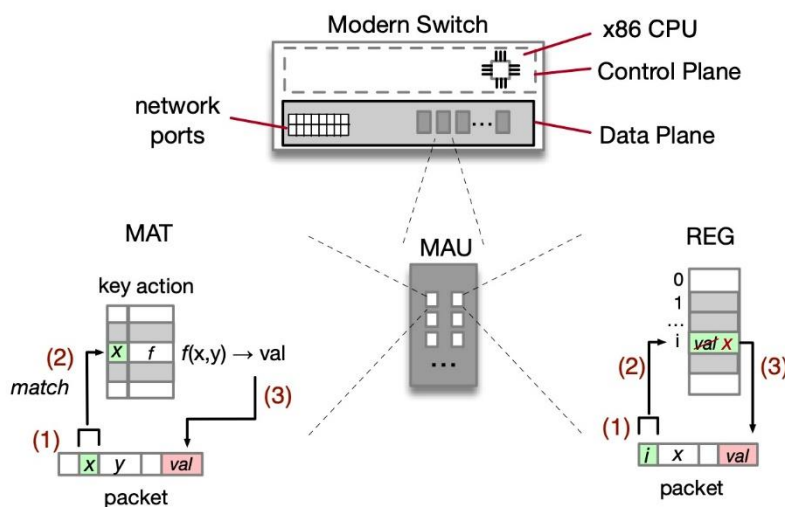


Figure 3-9: (Top) The switch is composed of a control plane and a data plane. The data plane has a pipeline of Match-Action Units (MAUs) with two types of storage: Match-Action Tables (MATs) and Registers (REGs). (Bottom) A MAT can read and alter a packet by: (1) selecting the field(s) to match; (2) performing the match, e.g., via equality comparison, and if an entry is found; (3) executing the matched entry's action, altering the packet's contents. A register works similarly, although the access to registers is positional.

Each MAU can locally implement an abstraction called Match-Action Tables (MATs). A MAT is where packet processing takes place. It can execute a function consisting of a lookup operation (the match) against a locally stored table and the application of a side-effect (the action) associated with the matched value. Figure shows how the operation works.

MATs can be programmed by deciding (a) which packet value(s) to use in the lookup operation, (b) what calculation to perform as the action, and (c) where to store the result. For instance, a switch must decrement the TTL (time-to-live) field of an IP packet while forwarding it. It can recognize IP packets via the EtherType field. Therefore, the table would have an entry that would match when the EtherType is 0x0800, i.e., an IP packet. When matched, the table would trigger a TTL decrement. In Figure , MAT, x would be 0x0800, y would be TTL, and f would be $TTL - 1$. The TTL field would be overwritten with the new value in this example.

One important aspect of MATs is that their contents cannot be updated as part of an action. However, another construct called a Register (REG) allows updates. For instance, Figure (Bottom, Right) shows a register updating an entry using a value lifted from a packet and copying the old entry's value back into the packet.

A program on the switch consists of a sequence of match-action tables and registers configurations. Such programs can be written using P4, which is an open standard, or using proprietary languages such as Broadcom's NPL, Huawei's POF, or Xilinx's PX. As already mentioned, we focus on P4 in the following for its wide support.

The switch executes a program by moving each incoming packet through that program's MAU/REG pipeline, invoking all MATs and REGs along the way. Because packets can only move into one direction—the next stage on the pipeline—this computing paradigm is sometimes called feed-forward model.

One interesting property of commercial programmable switches is that they impose a strict pipelining discipline, i.e., each MAU/REG takes the same amount of time with every packet. This is possible because the P4 compiler can cap the actions' length to a given maximum set of steps. Programs with actions of longer durations simply fail to compile. This uniformity allows the switch to move all packets traversing the switch to their respective next MAU/REG in lock-step. In other words, forwarding a packet through a pipeline incurs latency but does not affect bandwidth. If a series of packets arrive at the maximum bandwidth (called line rate), they will be forwarded at the same rate. Therefore, the programs that compile successfully will handle packets at network speed.

Dedicated processing can be added or modified in a programmable switch simply by describing these protocols as a sequence of matches and actions. Researchers and developers quickly realized that this computing model could express logic beyond networking, allowing them to use the switch as an application platform. However, converting algorithms from general-purpose machines to this computing model requires a paradigm change. The computations now have to be described as side-effects of forwarding a packet.

3.4.6.3 Offloading Graph-Based Operations

The first generic operation decided to offload was graph-based operations, most specifically Graph Pattern Mining (GPM), which is an important class of graph analysis. It finds all subgraph occurrences that match specific patterns. We decided to start with offloading GPM operations for two reasons: i) because GPM operations are, we believe, a very good match with the (limited) expressivity of P4 and hence were a great choice as a first offloading experiment and ii) because GPM operations are prominent in many applications, including listing cliques, finding motifs, and in mining frequent subgraphs.

A common approach to execute GPM algorithms is to iterate over all possible subgraphs and check if they match the desired pattern. This is done using a two-step process. First, subgraph enumeration extends subgraphs by adding one more node. This generates intermediate candidate subgraphs. Second, pattern analysis examines these intermediate subgraphs and looks for the pattern. Successful candidates are the ones that meet the pattern's matching criteria. This approach is iterative; meaning that the successful subgraphs that passed the pattern analysis are then extended again by adding one more node, and so on. The process ends when no further subgraphs can be extended.

There are two techniques in the literature to guarantee that each subgraph result is generated exactly once. First, techniques to discard the generated duplicate results. Second, there are techniques to avoid enumerating duplicate subgraphs. This reduces the enumeration search space in GPM problems. For example, converting the input graph to a Directed Acyclic Graph has been widely adopted.

In typical graph analysis problems, the size of the input graph is problematic. However, the main challenge in graph pattern mining is the explosion of intermediate results. The enumeration process is typically computationally intensive. As discussed above, it involves enumerating all candidate subgraphs and testing them for patterns. Even with a relatively small input graph, a huge intermediate state is generated. Figure 3-10 illustrates this challenge. The figure shows the number of results generated when executing the cliques listing task for sizes 3, 4, and 5 for three datasets. The plots show the number of generated cliques as well as the number of generated intermediate subgraphs. We see two main observations. First, the number of cliques and subgraphs increases as the pattern size increases. Second, the number of generated subgraphs is larger than the filtered out cliques results.

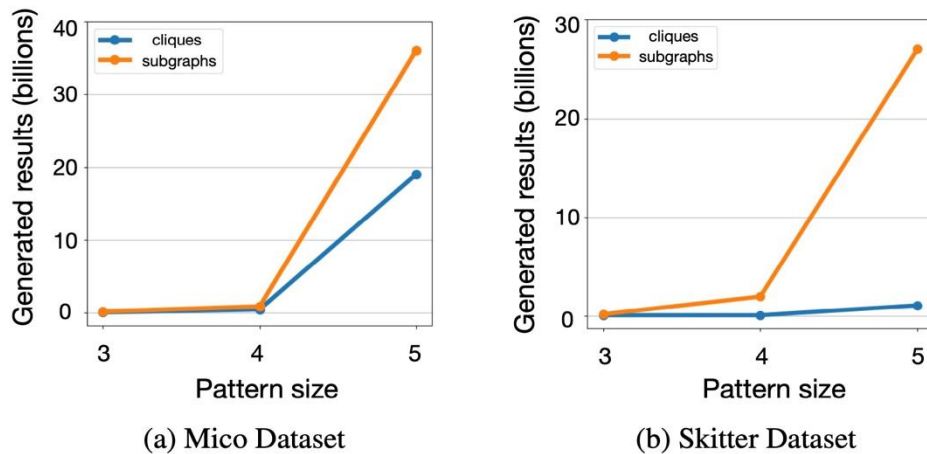


Figure 3-10: Intermediate results generated from GPM computation on two well-known graphs

Another challenge in GPM tasks is the imbalanced workload that creates computational bottlenecks. This is due to the heavily skewed degree distributions of real-world graphs. Only a few nodes are highly connected while the majority of nodes have very few neighbors. As a result, the majority of generated subgraphs contain these few high-degree nodes. For example, when running the 5-cliques task on the Mico [Hussein23] dataset, we find that a few high-degree nodes (0.1% of the graph nodes) are present in ~50% of the generated subgraphs. If left unhandled, resources responsible for processing the high degree nodes can become over consumed. Therefore, achieving good resource utilization becomes a challenging task.

That is why we designed and implemented a framework to offload the whole GPM process, or only the skewed, problematic portion of the problem, to a powerful P4 node. The overall idea behind the offloading is illustrated in Figure 3-11 below. Numbers in white illustrate the process from the edge nodes perspective: (1) a node syncs with the orchestrator (in this case running on the control plane) and solicits some work. (2) The requesting node is assigned a fragment of the problem, and (3) starts processing that fragment. (4) The resulting patterns are output by the nodes. The workflow on the accelerator, in red, is somewhat similar, even though the individual steps are performed differently: (1) the switch registers to the orchestrator to announce that it can accelerate part of the process. (2) the switch accelerator is assigned specific graph fragments (3) The nodes send the corresponding data (a subgraph) by D-RDMA to the accelerator (4) The subgraph is handled on the P4-accelerated node and the desired patterns are emitted as results.

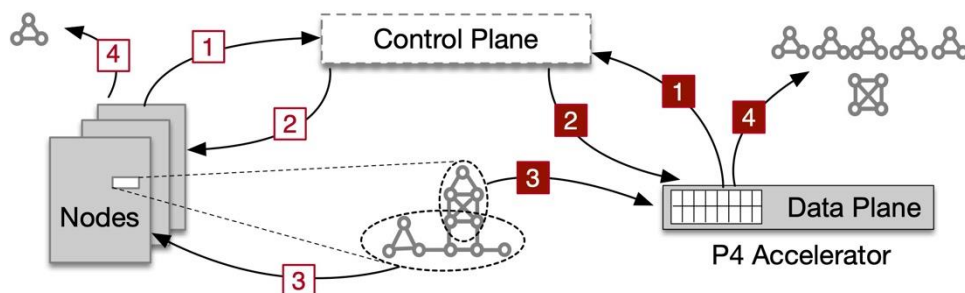


Figure 3-11: Our offloading framework main workflow. The edge nodes and a P4 accelerator (in this case a powerful and programmable P4 switch) operate independently but can offload GPM computations dynamically. In red, part of the problem is offloaded to the powerful P4 accelerator: (1) Resources or the P4 switch's data plane become available and (2) the switch accelerator is assigned specific graph fragments (3) The nodes send the corresponding data (a subgraph) by D-RDMA (4) The subgraph is handled on the P4-accelerated node and the desired patterns are emitted as a result.

While conceptually simple, the whole process is actually technically complex, and was the subject of full research paper presented at SIGMOD (the top venue for data-intensive systems) in 2023 (see [Hussein23] for details). This complexity stems from two points: first, while we picked a relatively simple task (GPM) to offload first, the programming model of P4 is limited and reformulating GPM in a P4-compatible, feed-forward fashion was technically challenging. Second, the P4 switch we used is actually very powerful, and optimizing the overall offloading process to take full advantage of the accelerator was technically challenging. The results are however extremely promising: we outperform the state of the art (Fractal [Dias2019]) by more than an order of magnitude (i.e., more than 1000%) on average, as Figure 3-12 below illustrates.

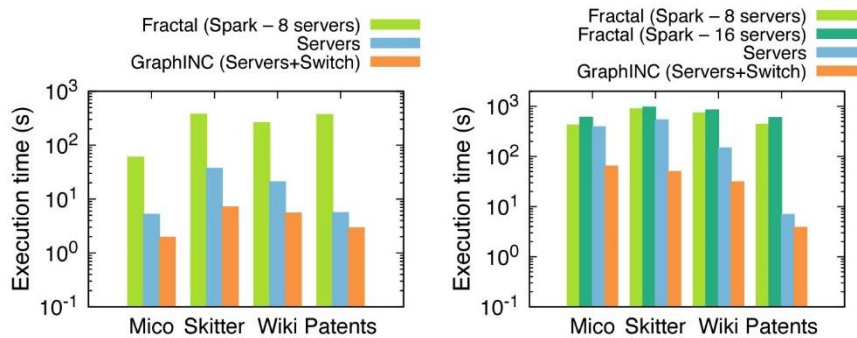


Figure 3-12: Comparison of running GPM task entirely on nodes (servers) using a state-of-the-art framework (Fractal [Dias2019]) versus offloading the workload to the switch using our framework called GraphINC.

3.4.7 Offloading Complex SmartEdge Operations Using SPARQL

Following the first generic offloading framework centered around GPM computation presented above, we plan to focus our efforts on offloading a wide range of operations in SmartEdge focusing on SPARQL. SPARQL is a very generic language, that is heavily used to describe operations as the rest of this document showcases. Offloading SPARQL operations to P4-accelerated devices could considerably accelerate the SmartEdge workload, but is technically challenging since SPARQL is a full-fledged, expressive language with many different and powerful operators.

Hence, we plan to focus on accelerating a subset of SPARQL in P4 in the rest of the project, focusing on the following important operations for SmartEdge:

1. Conjunction and disjunction of triple patterns
2. Property paths using advanced path expressions supported by SPARQL 1.1 (e.g., *AlternativePath* or *ZeroOrMorePath*)
3. Complex aggregate operations supporting *GroupBy* and *Having*.

We plan to generalize the design that we took for our first offloading operation (illustrated in Figure) for those three points. In terms of techniques, point 2. can hopefully be based on part of our GPM framework (since both GPM operations and advanced path operations require to

explore parts of graphs in an iterative manner). For point 1., we plan to extend our initial, previous work on offloading relational operators [Lerner2019]. We will need to design an entirely new technical solution for point 3. The design and implementation of this offloading solution will be a collaboration between FRIB and TUB taking place in 2024-2025.

3.4.8 Runtime Optimizer

Finally, we plan to build a component running on the SmartEdge orchestrator to optimize the offloading process. The overall design of this runtime optimizer is given below in Figure 3-13.

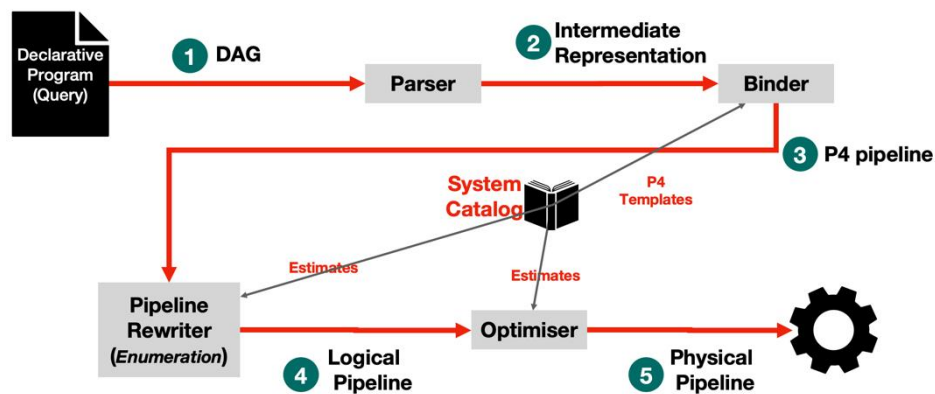


Figure 3-13: The design of our runtime optimizer for offloading operations in SmartEdge; as all components from this task, the optimizer will take a declarative specification of complex operations to offload (1), will translate this high-level representation of the complex operation into some intermediate representation (2) that can be translated into P4 and optimized before instantiating the optimized pipeline that needs to be run on a one or several accelerated node.

As all other components designed in the context of this task, the optimizer uses high-level, low-code declarative constructs as input, more specifically a declarative program describing the complex operations to offload. The optimizer then translates this program into a Directed Acyclic Graph of operations, that are then translated into some intermediate representation to be optimized. The optimization process itself will leverage a system catalog containing set of rules to rewrite and optimize the offloading operation into an efficient physical pipeline that can be run on one or several P4-accelerated nodes (in a way similar to the optimization of complex programs in data-intensive systems). The optimization process will span three different layers, as illustrated in Figure 3-14: 1. physical optimizations (e.g., based on statistics to pick the most efficient physical plan to run the complex offloading), 2. logical optimization (e.g., based on logical rules on how to move various operators to obtain more efficient offloading plans) and 3. hardware-specific optimizations (e.g., fusion of operators that can be jointly run in a common P4 pipeline, for example fusing a join and a group-by operation).

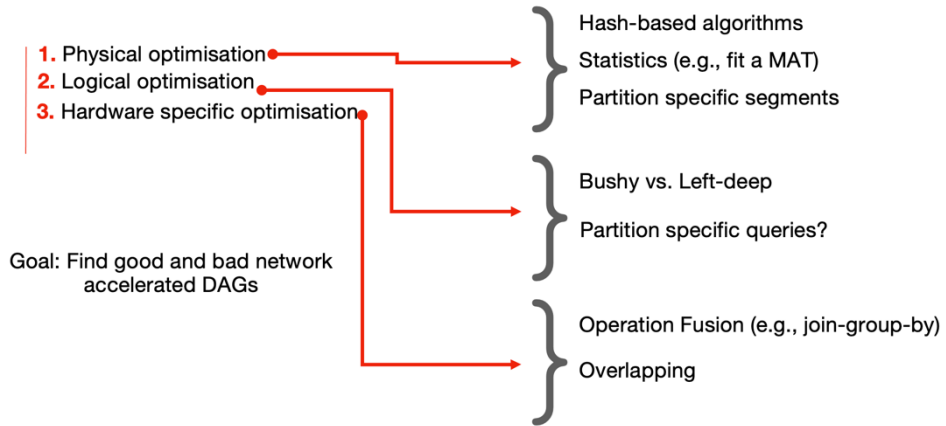


Figure 3-14: the three layers of optimization that will be supported by our runtime optimizer.

4 ADAPTIVE COORDINATION AND OPTIMIZATION MECHANISMS

This section delineates the ongoing work regarding the design and implementation of the adaptive coordination and optimization mechanism for SmartEdge swarms, focusing specifically on our progress in Task T5.3.

The content of this section is structured as follows:

- Section 4.1 presents the overview of Task T5.3, explaining how it interfaces with components developed in other tasks.
- Section 4.2 summarizes the relevant requirements for Task T5.3, as defined by the use case owner from WP2. This ensures that our developments align seamlessly with the envisioned functionalities and goals.
- Section 4.3 presents the preliminary works that will serve as the foundation for realizing Key Performance Indicators (KPIs).
- Section 4.4 delves into a detailed discussion on the architecture of the coordinator and optimizer. These essential components, to be developed in Task T5.3, play a pivotal role in achieving adaptive coordination and optimization within SmartEdge swarms.

4.1 OVERVIEW OF SWARM COORDINATION AND OPTIMIZATION

Task 5.3 aims to enable the autonomous coordination of a swarm at runtime and enhance its ability to adapt to changes in its surrounding environment. The objective is to develop mechanisms that enable adaptive coordination and optimization specifically tailored for SmartEdge smart nodes within the SmartEdge swarm. This mechanism empowers the swarm to self-manage and self-coordinate, enabling it to efficiently respond to varying environmental conditions. By doing so, it ensures that the SmartEdge smart nodes can seamlessly adapt and optimize their operations in real-time, thereby enhancing the overall performance and responsiveness of the swarm in dynamic and evolving scenarios.

To establish its operational functionality, the development of T5.3 relies on the runtime toolchain infrastructure established in task T5.4. During the initial phase, a compiler developed within T5.4 to parse high-level application specifications to construct the processing pipeline. This complex process involves systematically decomposing the application logic into a structured set of tasks, each accompanied by its corresponding performance requirements. The processing pipeline is conceptualized as a semantic program, meticulously implemented using CQELS-RL, as detailed in Section 5.4.1. Concurrently, within this phase, the preparation of runtimes for SmartEdge primitives (as outlined in Section 5.4.3), necessary for executing tasks in the processing pipeline, is undertaken. Subsequently, these runtimes will be deployed onto SmartEdge nodes when forming a swarm.

To deploy the application described in the semantic program, the implementation of T5.3 identifies SmartEdge nodes and specifies the SmartEdge primitives to be set up on each node. It then orchestrates their integration into a cohesive swarm. Each node within this swarm is meticulously equipped with the necessary capabilities, precisely aligning with the specific demands outlined in the application specifications. This dynamic formation of nodes ensures the successful execution of tasks in a distributed manner, leveraging the strengths of each node to collectively achieve the high-level application objectives.

The autonomous coordination mechanism implemented within Task T5.3 also adapts the swarm's operation to changes in the surrounding environment. For instance, it dynamically

reconfigures the execution plan when a node joins or exits the swarm. Furthermore, it may request additional nodes to join and share the workload during periods of increased demand. Additionally, Task T5.3 develops an adaptive optimization mechanism to empower SmartEdge smart nodes in real-time execution adjustments, aligning with predefined optimization goals. These goals encompass critical aspects such as energy consumption, bandwidth utilization, latency, and computing resources. SmartEdge nodes utilize the optimizer to make intelligent resource allocation decisions. This includes determining the most efficient use of computing capabilities, deciding whether to execute tasks locally, distributing the workload among swarm nodes, or even offloading parts of some tasks to the cloud (with the help of the mechanisms described in the previous section in the context of Task 5.2).

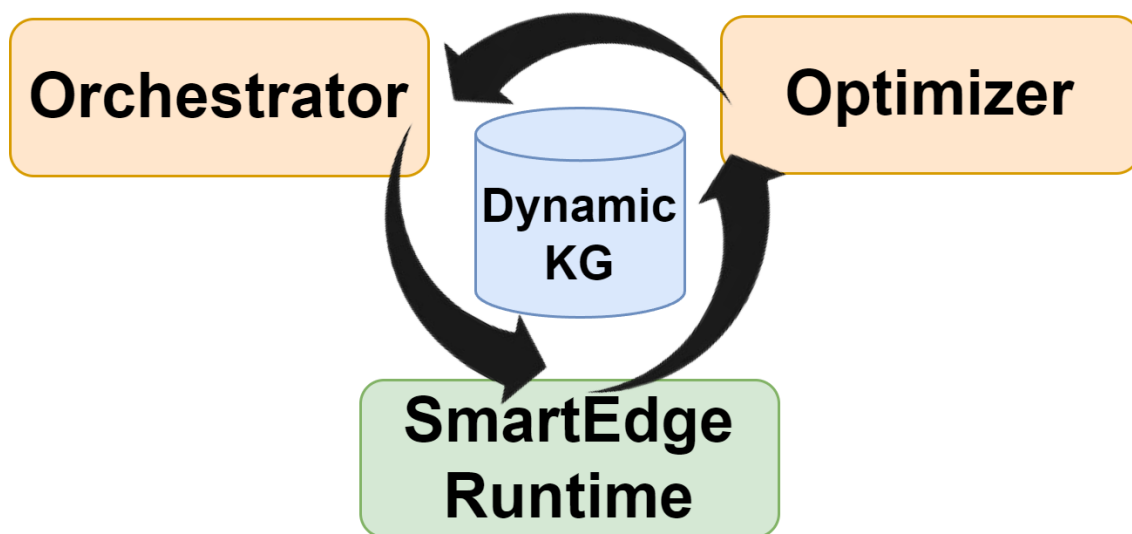


Figure 4-1. Architectural overview of SmartEdge Orchestrator and Optimizer

Figure 4-1 provides an overview architecture of the components that enable the coordination and optimization mechanisms within the SmartEdge swarm. In normal setup, a Swarm Coordinator can have both an Orchestrator and an Optimizer like a DKG. At the core, an Orchestrator is responsible for orchestrating collaborative actions of a set of SmartEdge smart nodes. When a semantic program is received by a smart node, the Orchestrator decomposes sub-tasks to construct a federated execution plan. This plan intricately links various data sources to specific processing operators and determines where the resulting processed data should be directed. Importantly, the Orchestrator possesses the intelligence to adapt dynamically to changing circumstances. If a single smart edge node lacks the computational resources to handle the entirety of a given task, the Orchestrator intelligently partitions the task into smaller segments (whenever possible) and assigns them to other capable smart nodes within the swarm.

To assist the coordinator in locating data sources and other nodes, a Dynamic Knowledge Graph (DKG) is paired with it. Functioning as a real-time repository, the DKG facilitates data source and node discovery within the SmartEdge environment. By maintaining up-to-date information on available smart nodes, including hardware specifications and resource availability, the DKG empowers the Orchestrator to make informed decisions. Through semantic annotation, this information is structured to enable precise matching of task requirements with individual node

capabilities. This fusion of dynamic coordination and intelligent resource allocation underpins the operational efficiency of the SmartEdge swarm. The DKG can be centralized on a single smart node or distributed across multiple smart nodes. For distributed DFG, we provide a detailed description of the semantic-based discovery and formation in Section 4.3.2, alongside the presentation of an initial version of such a distributed Knowledge Graph for edge devices.

The Optimizer continuously monitors smart node operations, available resources, and program performance to assist the federator in fine-tuning the execution plan. For instance, it predicts if a smart node is about to leave or become unavailable for its assigned swarm. By analyzing factors such as coordination and vehicle speeds, it can anticipate when a car will depart a junction and join the next, allowing the federator to prepare accordingly. Similarly, based on current resource availability, such as battery level, the optimizer predicts if a node can no longer fully participate in the federation, prompting the relocation of the running program to another node.

Moreover, the optimizer learns performance metrics to optimize swarm efficiency. For instance, if an object detection task on a Jetson device does not require real-time processing, it can suggest transitioning the node to a low-power consumption mode to conserve energy. Conversely, if a Jetson device cannot perform the task quickly enough, the optimizer can distribute the task to other devices. Through continuous learning, the optimizer determines the most effective topology for task distribution, ensuring optimal swarm performance. When a task is offloaded to a special node with hardware acceleration, the optimizer calls the runtime optimizer of T5.2 (see above) to optimize the hardware acceleration part of the operation.

Given the prevalence of streaming data in SmartEdge applications, Section 0 conducts an in-depth exploration of the mechanisms involved in distributing semantic stream processing tasks among participants. This analysis sheds light on how different network topologies influence the performance of streaming systems, providing valuable insights for optimizing real-time data processing in dynamic environments.

Furthermore, the performance of SmartEdge applications hinges significantly on the efficiency of communication technologies. In Section 4.3.1, the focus shifts to examining the latency implications of various implementations of the Data Distribution Service (DDS) protocol. As a fundamental communication protocol used extensively in SmartEdge scenarios, understanding the latency characteristics of different DDS implementations is crucial for ensuring timely and reliable data exchange between devices in the SmartEdge ecosystem.

4.2 REQUIREMENTS

ID/Ver: SW-002/v1.1	Related Use Case(s): UC-1, UC-3, UC-5	Task: T5.3
A swarm smart-node is an autonomous entity that can actively participate in a swarm and must be able to support its basic functionality. For example, the swarm smart-node must have the processing environments to be able to host the SmartEdge swarm components.		
T5.3 provides a discovery mechanism to find the proper smart nodes for specific tasks.		
ID/Ver: SW-003/v1.1	Related Use Case(s): UC-2, UC-3, UC-5	Task: T5.3
A swarm coordinator is a swarm smart-node that must support the advanced functionality necessary to form and maintain a swarm. For example, the ability to onboard or offboard nodes to and from the swarm, or in UC5 the provide medical staff with such node that support		

coordination, holistic management, and monitoring of a group of patients represented by their swarm nodes.

T5.3 provides smart nodes with the ability to form and coordinate a swarm via a Federator.

ID/Ver: SW-004/v1.1	Related Use Case(s): UC-1, UC-2, UC-3, UC-4, UC-5	Task: T5.3
----------------------------	--	-------------------

A manifest swarm must consist of one or more swarm smart-nodes and at least one of those smart-nodes must be a swarm coordinator.

ID/Ver: SW-005/v1.1	Related Use Case(s): UC-3	Task: T5.3
----------------------------	----------------------------------	-------------------

A swarm should have three or more swarm coordinators to provide resilience if a swarm controller is lost.

ID/Ver: SW-006/v1.1	Related Use Case(s): UC-3, UC-5	Task: T5.3
----------------------------	--	-------------------

A swarm must have an odd number of swarm coordinators to prevent the “split brain” scenario.

T5.3 will ensure that during swarm formation, at least three smart-nodes will be discovered and serve the swarm as the swarm coordinators.

ID/Ver: SW-007/v1.1	Related Use Case(s): UC-2, UC-3	Task: T5.3
----------------------------	--	-------------------

The swarm coordinator(s) must maintain the state of the swarm. For example, which nodes are part of the swarm.

T5.3 will ensure that the coordinator maintains a DKG, which can be implemented as an RDF store storing the metadata of all the nodes serving the swarm.

ID/Ver: SW-008/v1.1	Related Use Case(s): UC-3	Task: T5.3
----------------------------	----------------------------------	-------------------

Swarm smart-nodes must reach a consensus on any action that affects the swarm. For example, the swarm smart-nodes must agree on the offboarding of a new node from the swarm. If a smart-node (A) requests to leave the swarm, but another smart-node (B) is relying on it to perform some operation, then smart-node (B) must have the opportunity to request smart-node (A) to remain in the swarm. There may be exceptions to this rule, e.g., if all swarm coordinators agree to eject a node (C) from the swarm, then node (C) cannot block this action.

T5.3 will ensure such consensus, where the coordinator only agrees for a node to leave if a replacement is found and joins the swarm.

ID/Ver: SW-009/v1.1	Related Use Case(s): UC-3	Task: T5.3
----------------------------	----------------------------------	-------------------

Any swarm smart-node can abstain on any action proposed by another swarm smart-node. This may happen if the action does not directly affect the smart-node. Abstaining on proposed actions is desirable, as it reduces the decision-making process and possible action contentions.

T5.3 implement a Task Validation component to check if an assigned task will be executed or rejected. The decision will be based on task priority and resource consumption.

ID/Ver: SW-010/v1.1	Related Use Case(s): UC-3	Task: T5.3
----------------------------	----------------------------------	-------------------

The swarm must have a process to break any action contentions automatically between swarm smart-nodes, even if this is a random decision. Otherwise, external intervention would

be required, which goes against the concept of a swarm. Taking random actions can be an effective mechanism to break out of repetitive behaviour that continually fails.

The Task Validator developed in T5.3 will reject a task if it has the potential to cause action contentions.

ID/Ver: SW-011/v1.2	Related Use Case(s): UC-3	Task: T5.3
A swarm smart-node (A) must have the right to leave a swarm if it chooses, but another swarm smart-node (B), can request it to remain if it needs the assistance of smart-node (A). Smart-node (A) must then decide if it stays or leaves the swarm.		
To avoid conflict with SW-008, a smart-node will be programmed to only decide to leave if the coordinator finds a replacement or if it finds a replacement itself.		

ID/Ver: SW-013/v1.1	Related Use Case(s): UC-3	Task: T5.3
A swarm coordinator should only belong to one swarm at a time.		
T5.3 Task Validation will ensure that the coordinator does not process any tasks from other swarms		

ID/Ver: SW-021/v1.1	Related Use Case(s): UC-2	Task: T5.3
Swarm must always know what units; data sources and capabilities are available. This is similar to SW-007, however, swarm unit might still be present just unable to operate at full capacity, so this is a bit wider concept.		
This requirement will be developed in the next deliverable.		

ID/Ver: SW-024/v1.2	Related Use Case(s): UC-3, UC-4	Task: T5.3
The swarm Federated is the engine that executes a queries/DSL-based workload (e.g primitive recipe) at runtime. It has to check that all capabilities required for the swarm functionality, defined in a query, are available and can be executed.		
The T5.3 Task Validation module will fulfill this requirement.		

ID/Ver: SW-026/v1.1	Related Use Case(s): UC-3	Task: T5.3
In many cases the swarm coordinator and orchestrator are coresident on the same swarm smart-node, but they can reside on different nodes.		
T5.3 will enable the coordination and orchestration in any smart node. (Section 5.4.1)		

ID/Ver: SC-002/v1.1	Related Use Case(s): UC-3	Task: T5.3
SmartEdge must provide a procedure so that a Swarm can communicate events about its state to swarm participants that registered for certain swarm events.		
Coordinators of the swarm will be allowed to update the state of the swarm to the DKG.		

ID/Ver: SC-003/v1.1	Related Use Case(s): UC-3	Task: T5.3
----------------------------	----------------------------------	-------------------

It must be possible to register for application specific events that can be published by SmartEdge applications. The events must support application specific data if the application needs to communicate payload data.

ID/Ver: SC-004/v1.1	Related Use Case(s): UC-3	Task: T5.3
----------------------------	----------------------------------	-------------------

It must be possible to set a time to live window until when an event is valid. After that it is considered outdated and can be removed.

ID/Ver: SC-005/v1.2	Related Use Case(s): UC-3	Task: T5.3
----------------------------	----------------------------------	-------------------

Events that are propagated by a swarm node should have a unique name space and ID so they will not conflict with application specific events.

ID/Ver: SC-006/v1.1	Related Use Case(s): UC-3	Task: T5.3
----------------------------	----------------------------------	-------------------

Events should contain data about the creation source, time, and date.

ID/Ver: SC-007/v1.1	Related Use Case(s): UC-3	Task: T5.3
----------------------------	----------------------------------	-------------------

SmartEdge swarm events shall be only created by the swarm itself (and not by any application that uses SmartEdge).

ID/Ver: SC-019/v1.2	Related Use Case(s): UC-3	Task: T5.3
----------------------------	----------------------------------	-------------------

There has to be sanity checking of data coming from the streams. In particular, in UC-2 the long-term solution is to allow “outsiders” to send data from V2X capable vehicles, this has to be checked. In UC-5 each sensor data shall be checked against unique sensor properties set during the configuration by the authorised swarm admin.

ID/Ver: SC-022/v1.2	Related Use Case(s): UC-3	Task: T5.3
----------------------------	----------------------------------	-------------------

A protocol must be implemented for smart nodes in a swarm to exchange high-level semantic information about their environment.

T5.3 will create an output handler to package output data into semantic messages, which will include the above metadata.

ID/Ver: AL-001/v1.1	Related Use Case(s): UC-2, UC-3, UC-5	Task: T5.3
----------------------------	--	-------------------

A swarm should be able to run more than one application at a time, providing they do not directly compete.

A smart node can execute more than one task depending on its resource availability.

ID/Ver: AL-002/v1.1	Related Use Case(s): UC-2, UC-3, UC-5	Task: T5.3 & T5.4
----------------------------	--	--------------------------

Running an application should be coordinated by a designated swarm smart-node, called the orchestrator, but it may enlist other swarm smart-nodes into performing the task described by the application. For example, an AMR can only hold one product at a time, it needs to enlist a mobile rack to hold multiple products of this type. As this is also the functional objective of the mobile rack, there is no conflict of interest.

T5.3 enables the orchestration ability for smart nodes, and the Semantic Program model allows such application logic to be understandable to smart nodes.

ID/Ver: AL-003/v1.1	Related Use Case(s): UC-2, UC-3, UC-5	Task: T5.3
----------------------------	--	-------------------

Technically a swarm should have at least one running application to continue to exist. If the swarm has no running application, it has no purpose and so should be dissolved by agreement from its constituent swarm devices. In practice there may be a small time lag between the swarm completing its last task and starting the next. If the swarm dissolved immediately it might have to be reformed a little time later, which could incur an operational penalty. Instead, the swarm should exhibit some level of “stickiness” for a time, i.e., the swarm may continue to exist without an immediate purpose.

If no application is running on a swarm, T5.3 does not dismiss the swarm, but allows a node to serve another swarm if requested.

ID/Ver: AL-004/v1.1	Related Use Case(s): UC-2, UC-3, UC-5	Task: T5.3
If an application orchestration smart-node fails or leaves the swarm in the middle of the execution of an application, if possible, another smart-node in the swarm should take over as orchestrator.		
See SW-004, SW-005, SW-006 and SW-008		

ID/Ver: AL-005/v1.1	Related Use Case(s): UC-2, UC-3, UC-5	Task: T5.3
A swarm device may refuse to cooperate with an application orchestration device if this conflicts with a higher objective. For example, an AMR may enlist a mobile rack to take another product, but the rack may refuse if the product is of the wrong type, or the rack is full. In this case the AMR should recruit a new device (new rack) into the swarm so the task can be fulfilled.		
The Task Validator developed in T5.3 will reject a task if it has the potential to cause a conflict with a higher objective.		

ID/Ver: AL-006/v1.1	Related Use Case(s): UC-2, UC-3, UC-5	Task: T5.3
It may be possible to have more than one orchestrator running a different application in the same swarm at the same time.		
See AL-001.		

ID/Ver: AL-007/v1.2	Related Use Case(s): UC-2, UC-3, UC-5	Task: T5.3
An orchestrator may have more than one task assigned at a time, but only one application task should run at a time. Therefore, the orchestrator should implement a task queue. The orchestrator will run the application task with the highest priority. If this application can't run because the prerequisites have not been met, then it runs the next highest priority application task. For example, if the AMR can't perform the highest priority task, to move a product from the conveyer to a rack, because there are no more spaces in the rack; the AMR may take a lower priority task to move the rack to its next processing stage and enlist into the swarm a new empty rack.		
The Task Validator developed in T5.3 will reject a task if the smart node is busy.		

ID/Ver: AL-008/v1.2	Related Use Case(s): UC-2, UC-3, UC-5	Task: T5.3
----------------------------	--	-------------------

A swarm device may orchestrate its own tasks, whilst collaborating with another orchestrator on its tasks. A device should only participate in one task at a time, and so must put all tasks, its own and other orchestrators, into a single task queue. This is particularly important for battery powered AMRs, who's highest priority task is to ensure they have sufficient power to complete the current task and get to a recharging station. This is also a reason why swarm devices have the right to leave a swarm.

The federation mechanism developed in T5.3 will fulfill this requirement.

ID/Ver: AL-009/v1.2	Related Use Case(s): UC-2, UC-3, UC-5	Task: T5.3
Where possible if a task cannot be completed or the application fails part way through, e.g., the path of an AMR is blocked by an unexpected obstacle, the orchestrator should try to autonomously recover from the incident, e.g., replan an obstacle free path. If it can't recover, it should request manual intervention.		
Coordinator should ensure a task is complete		

ID/Ver: LC-004/v1.2	Related Use Case(s): UC-2, UC-3, UC-4	Task: T5.3
SmartEdge runtime should provide discovery functionality to discover available devices and their capabilities.		
T.53 will develop a SmartEdge node discovery supporting the Federator subsystem.		

ID/Ver: LC-018/v1.1	Related Use Case(s): UC-2, UC-5	Task: 5.3
Capability for receiving control messages from nodes and relaying them to appropriate participants. For example, we might have a process outside the SmartEdge connected to a node and receiving data, analysing it and creating events.		
T5.3 Will provide a message manager		

ID/Ver: LC-023/v2.1	Related Use Case(s): UC-4	Task: 5.3
A mechanism that allows to forecast the Swarm state must be provided.		
T5.3 implements a State Predictor to predict the state changes of a swarm.		

ID/Ver: CSI-021/v1.1	Related Use Case(s): UC-1, UC-2, UC-5	Task: T5.3
Data provided by continuous semantic data streams must be filtered through IDM, policies, and security requirements before the data is allowed to be used (e.g., by recipes) to prevent privacy violations.		
T5.3 will integrate the implementations of IDM into the Input Handler.		

4.3 PRELIMINARIES AND STATE OF THE ART

An examination of the current state of the art in distributed systems highlights three pivotal components in our context: DDS-based communication, P2P-based discovery and federations, and continuous query federation.

In DDS-based communication, our exploration delves into the intricacies of data transfer speed, particularly when employing different settings within the DDS protocol. The focus is on optimizing the efficiency and performance of SmartEdge swarms by comprehensively understanding how variations in DDS protocol configurations impact the speed of data transfer.

P2P-based Discovery and Federations play a crucial role in fostering seamless communication among edge devices. The coordinator manages information exchange and task synchronization, promoting collaboration in a decentralized manner. This approach enhances efficiency and responsiveness within the edge computing ecosystem, creating a foundation for effective communication and coordination.

Continuous Query Federation introduces a dynamic dimension to exploration, focusing on real-time adaptability and responsiveness. The coordinator continuously monitors the states of edge devices, adapting coordination strategies to changing conditions. This dynamic approach ensures that the swarm remains agile and can optimize its performance under evolving circumstances, such as fluctuations in workload or environmental factors.

The coordinator's responsibility includes managing information flow, ensuring seamless integration and effective collaboration between decentralized and centralized elements. Additionally, the investigation emphasizes scalability and performance optimization as key considerations, addressing the evolving needs and expanding capabilities of the edge computing ecosystem. The preliminary and state-of-the-art exploration sets the stage for a comprehensive understanding of these components and their implications in enhancing communication, coordination, and overall performance in distributed systems.

4.3.1 Data Distribution Service -based Communication

As the first-generation software platform to develop robot applications, Robot Operating System (ROS1) has offered an enormous ecosystem of control, sensor, and algorithmic packages and provided utilities, such as introspecting communication, monitoring process, and exchanging time-series transformation. ROS1 has been used for developing complex robot applications from research groups to small-scale production companies. With the rise of commercial production for real-time, secure, large scale and reliable robots, ROS1 failed to fulfill several requirements. The lack of an inherited security mechanism, single point of failure due to the master node failure, and inconsistent data delivery over lossy links like satellite or wifi make it really difficult for the expanding robotics community to move forward with ROS1. The second-generation Robot Operating System (ROS2) has been redesigned from scratch to mitigate all of the previously mentioned challenges. ROS2 comes with a huge suite of required tools for robot developers. To configure, introspect, visualize, simulate, log, launch, source management, distribution, build process, and debug, ROS2 provides a command line tool and a graphical tool. Data Distribution Service (DDS) is the middleware of the ROS2 ecosystem, and it is responsible for communication among all the components, such as nodes, network API, and message parser of ROS2.

The Data Distribution Service (DDS) standard specification introduced by the Object Management Group (OMG) facilitates a publish/subscribe model with Quality of Service (QoS) enabled for timely and dependable data propagation. Various transport configurations, safety measures, scalability, resilience, security features, and fault tolerance capabilities of DDS establish it as a suitable transport layer for distributed systems. It maintains a Data-Centric Publish-Subscribe (DCPS) model. The DDS domain is identified with a domain ID, allowing publishers and subscribers from the same or different applications to write or read data to or

from a topic using data writers or data readers with the same domain ID. The topic within DDS binds the publisher and the subscriber within the Global Data Space (GDS), which includes an inherent isolation mechanism. The QoS of DDS distinguishes the behavior of each message regarding discovery and communication. All the publishers and subscribers within a domain are known as domain participants.

Different vendor-specific DDS implementations have their specific way of establishing communication for rapid scaling and integrating various nodes running in disparate domains or geographically dispersed. These linking applications work as a bridge across the different domains to establish a system-of-systems architecture. Each bridging service has its own configuration, dependent libraries, and building process. There are different vendor-specific DDS implementations available, and they vary from each other by configuration, build requirement, packaging, and licensing but serve a common goal.

The following study aims to examine how increasing publisher frequency impacts the latency of three common ROS data types (Binary, String, and IMU) when nodes are connected via both wired and wireless communication, either within the same domain or across different domains. It focuses on UC3 and explores the potential application of UC2 if DDS is deployed in vehicles or RSUs. The experiment utilizes three types of machines acting as ROS2 nodes: Laptop computers, Raspberry Pi 3, and Raspberry Pi 4, each configured with three different DDS vendors as specified in Table 5.3.1 and utilizing both wired and wireless communication methods.

DDS Implementation	Bridging Application	Configuration
Eclipse Cyclone DDS	Zenoh-plugin	Publisher and Subscriber both require to have zenoh shared library
eProsima Fast-DDS	Integration Service	Configuration is done with the YAML file
RTI-Connex DDS	Routing Service	Configuration is done with XML file

Table 5.3.1 DDS implementations and Bridging applications.

4.3.1.1 Eclipse Cyclone DDS

Eclipse Cyclone DDS is one of the implementations of the OMG Data Distribution Service (DDS) specifications and DDSI specifications for seamless interoperability. Compared to other messaging solutions from Eclipse, Eclipse Cyclone DDS offers unparalleled data-sharing capabilities paramount to effective communication. Moreover, its data models come with fine-grained QoS properties, such as reliability, urgency, and persistence. These provide exceptional functional and non-functional properties, particularly for real-time or IoT systems that are time and mission/business critical.

Raspberry Pi3: In the graph shown in Figure 4-2, we can see how various frequencies used by the publisher affect latency. When wirelessly connected nodes using Eclipse Cyclone-DDS publish with different frequencies within the same domain, there is little difference in latency until the file size reaches 145KB.

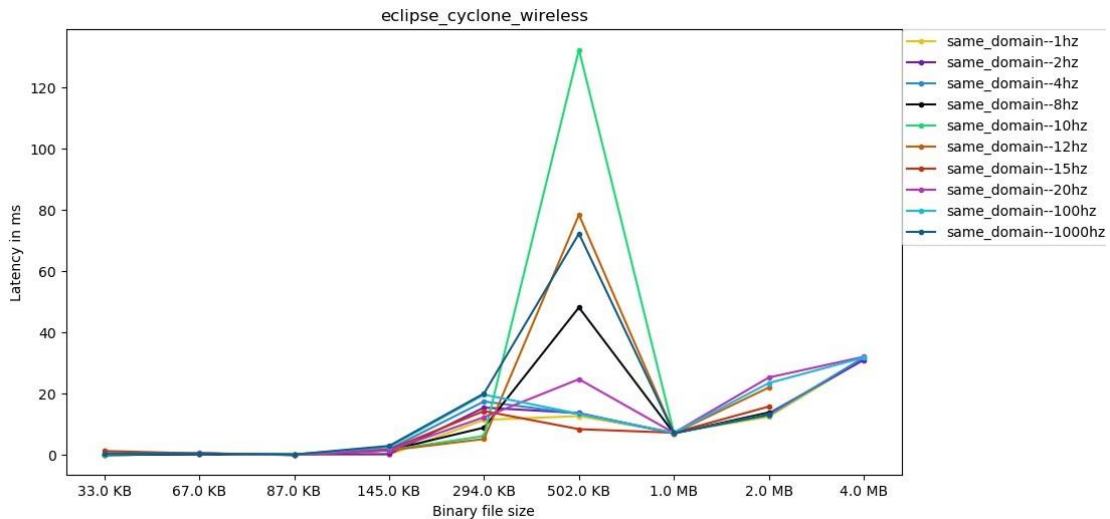


Figure 4-2. Publisher frequency impact on the latency using Eclipse cyclone DDS same domain wireless communication for Raspberry Pi3.

However, in Figure 4-3, when communicating across different domains, different frequencies result in varying latency for different file sizes, with higher frequencies causing increased latency for larger files. It is interesting to note that for both wirelessly connected nodes (same/different domain), there is a sudden increase in latency when the file size reaches 502KB.

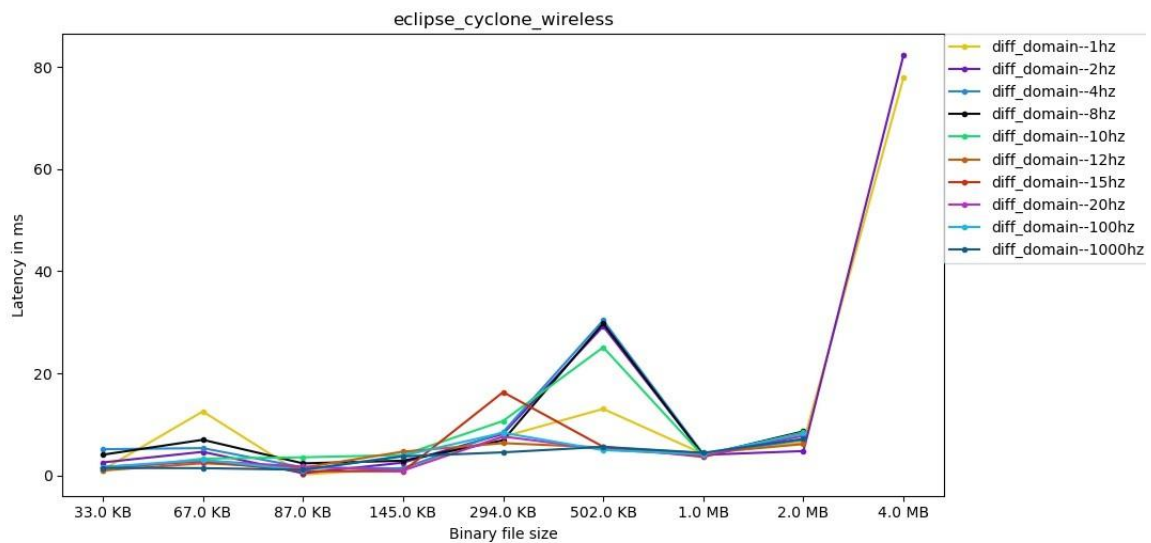


Figure 4-3. Publisher frequency impact on the latency using Eclipse cyclone DDS different domain wireless communication for Raspberry Pi3

When the file size exceeds 145KB, both wired and wireless same domain nodes experience similar latency increase characteristics. In same domain communication, the latency is higher for a 502KB file size than for a 1MB file size. However, different domain communication shows consistent latency characteristics across all file sizes in Figure 4-4; this is the opposite of wirelessly connected different domain communication.

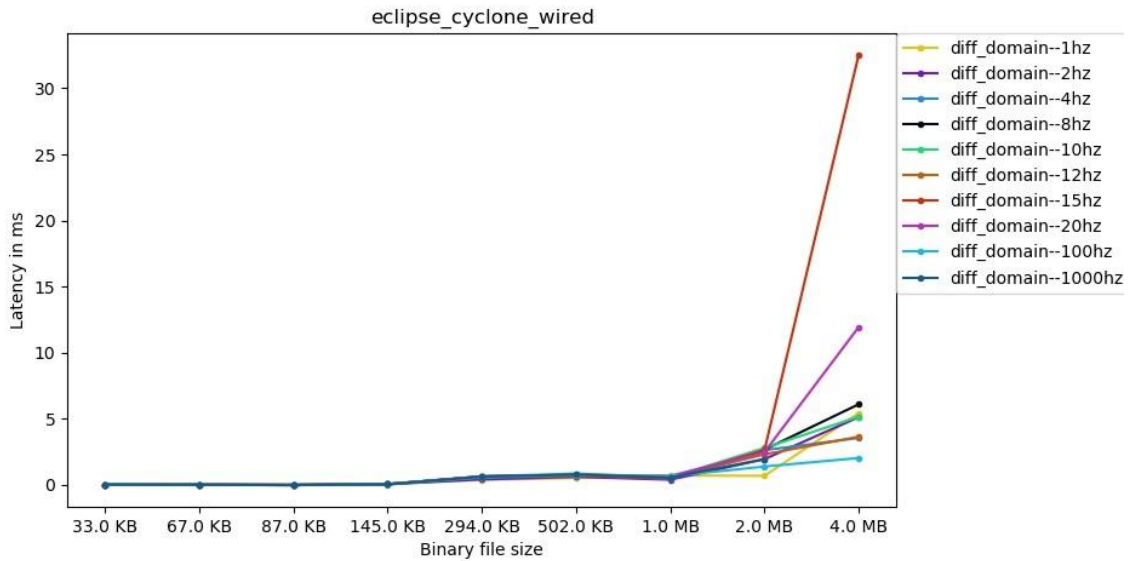


Figure 4-4. Publisher frequency impact on the latency using Eclipse cyclone DDS different domain wired communication for Raspberry Pi3.

Raspberry Pi4: The latency characteristics of Pi4 nodes wirelessly connected within the same domain are similar to those of Pi3 nodes for various frequencies when dealing with files up to 145KB in size. However, for files of 502KB, there is a sharp increase in latency, and for most frequencies, the latency drops when transferring with files of 1MB size. For the different domains wirelessly connected Pi4 nodes in Figure 4-5, latency varies for different frequencies. It is difficult to observe any pattern for the latency characteristics. Noticeably, for most frequencies, the 1MB file yields lower latency than the 502KB file size, similar to the same domain wireless communication.

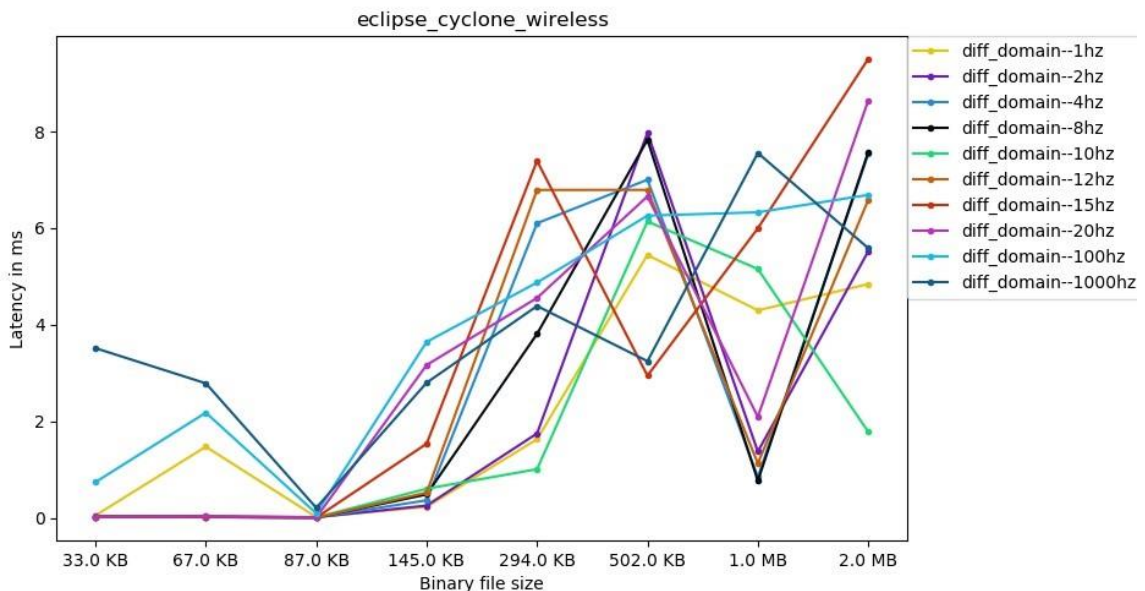


Figure 4-5. Publisher frequency impact on the latency using Eclipse cyclone DDS different domain wireless communication for Raspberry Pi4.

When connecting Pi4 nodes through wired connections within the same domain or across different domains, we observed that the latency feature appears to be consistent. Additionally,

we discovered that the file size (up to 145KB) is not affected by different publisher frequencies. Even with larger files (294KB and 502KB), different frequencies result in almost identical latency for both same domain and different domain connections Figure 4-6.

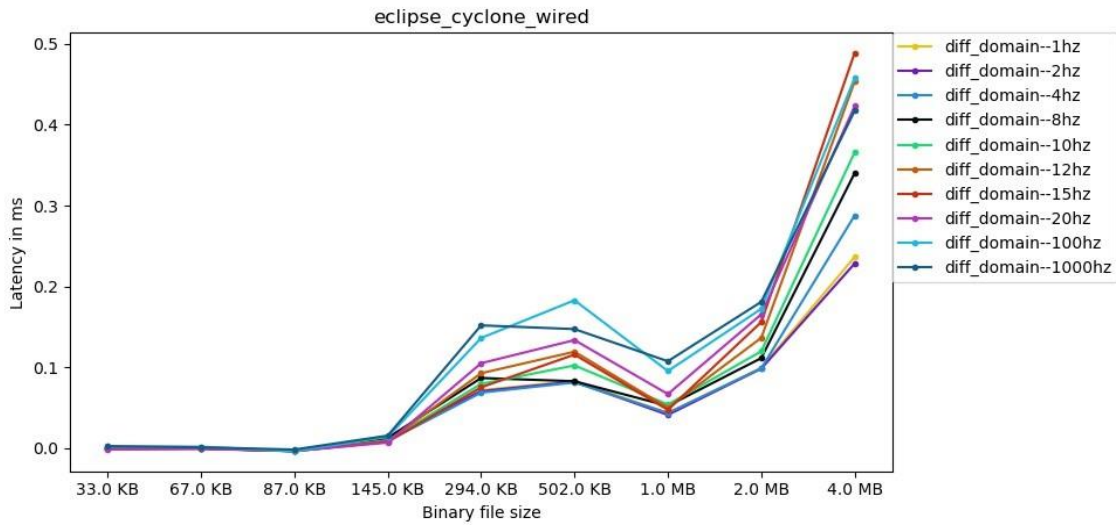


Figure 4-6. Publisher frequency impact on the latency using Eclipse cyclone DDS different domain wired communication for Raspberry Pi4.

Laptop Computer: When the same domain laptop nodes are connected wirelessly, all frequencies have similar latency for files up to 2MB. However, for Pi3 and Pi4, the same behavior is observed for files of size 145KB. A sudden increase in latency occurs when files reach 4MB at all frequencies. It is worth noting that the lowest publisher frequency, 1Hz, has the lowest latency for the largest file size for wirelessly connected laptop nodes of the same domain. When laptops from different domains are connected wirelessly, the different publisher frequencies have different latency characteristics for the same file size (see Figure 4-7). This is similar to the behavior observed in wireless communication between Pi3 and Pi4 devices from different domains. The graph demonstrates that lower frequencies have lower latency, even when transmitting larger files.

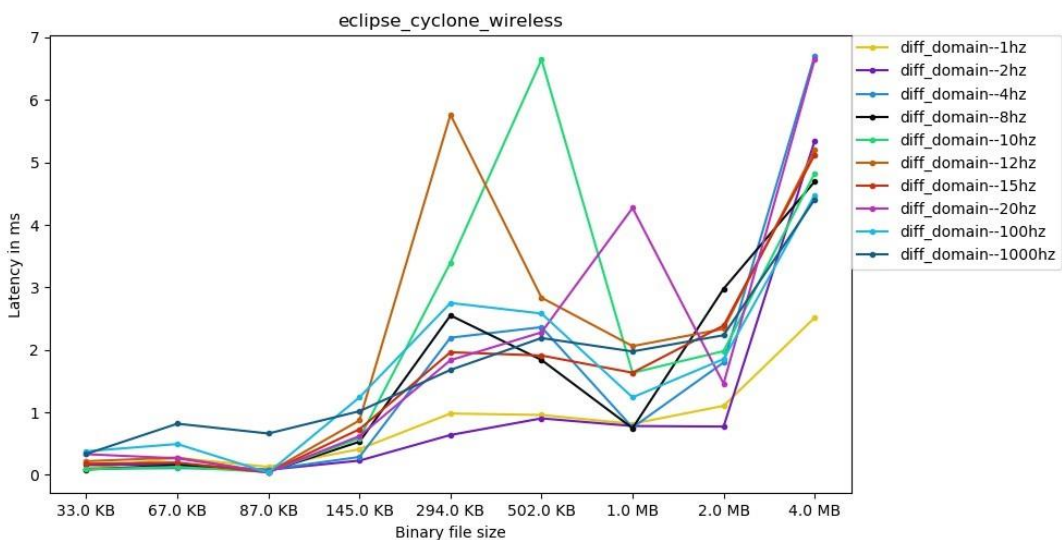


Figure 4-7. Publisher frequency impact on the latency using Eclipse cyclone DDS different domain wireless communication between laptop computers

4.3.1.2 eProsima DDS

The RTPS (Real Time Publish Subscribe Protocol) is a communication protocol that enables reliable pub-sub (publish-subscribe) communication over unreliable transports like UDP. It works for both unicast and multicast communication.

The OMG (Object Management Group) has standardized RTPS as the interoperability protocol for DDS implementations. It is a widely used standard for real-time applications in the aerospace and defense sectors. In addition to the RTPS implementations included in DDS implementations, independent, lightweight RTPS implementations are also available. The most prominent of these is eProsima Fast RTPS, which boasts excellent performance, features, and compliance with the most up-to-date RTPS standard, which is also known as eProsima DDS.

Raspberry Pi3: When Pi3 nodes are connected wirelessly within the same domain using eProsima DDS, they exhibit consistent latency for all frequencies, except for 1Hz when the file size is up to 294KB. However, when the file size is 502KB, there is a sudden increase in latency. For the 1Hz publisher frequency, the highest latency occurs for 294KB files, which is not typical of the other experiments. When the file size is 502KB, the 12Hz and 15Hz frequencies exhibit higher latency than the 1MB file. Latency decreases for all publisher frequencies once the file size reaches 1MB.

Our findings indicate that when Pi3 nodes are linked via wireless communication across various domains and use eProsima DDS to communicate, it results in a consistent latency effect across all frequencies and file sizes, except for the 1Hz publisher frequency, which affects 294KB, 502KB, and 1MB file sizes differently in Figure 4-8. This is an unexpected finding. To illustrate the overlapping lines, we have included a zoomed-in version of the latency diagram for 87KB and 67KB.

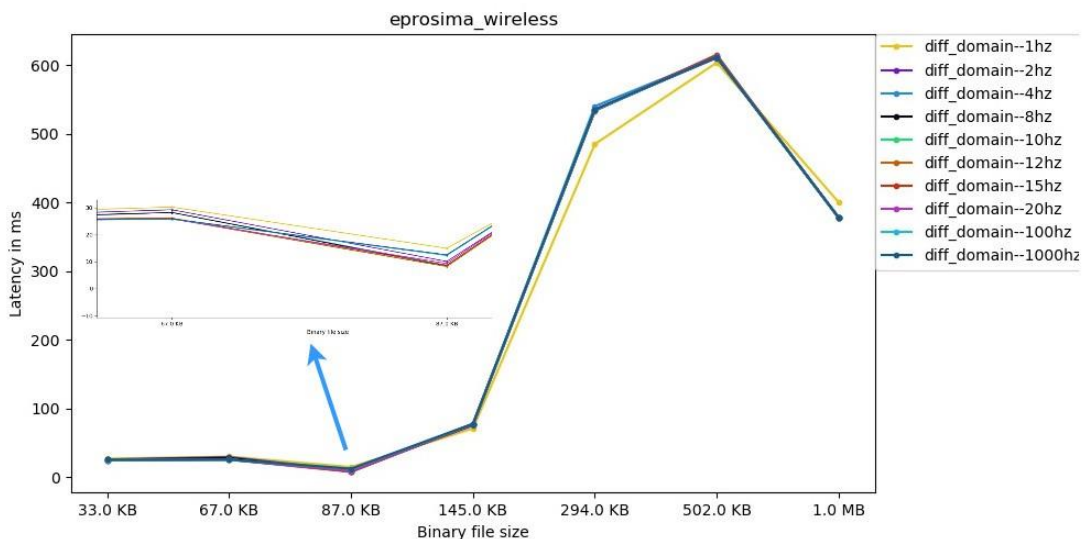


Figure 4-8. Publisher frequency impact on the latency using eProsima DDS different domain wireless communication between Pi3

Raspberry Pi4: Pi4 nodes that are connected wirelessly within the same domain and utilize eProsima DDS exhibit similar latency behavior as Eclipse Cyclone DDS at various frequencies. Latency remains consistent across all frequencies up to a file size of 145KB, but increases as file size increases, which is expected. Similar to Eclipse Cyclone DDS and RTI Connex DDS, eProsima

DDS results in the highest latency for 502KB files on Pi4 in Figure 4-9. Similar to Pi3 RTI Connexx Pi3 wired experiments in different domains and eProxima DDS Pi3 wired both the same and different domain, we noticed a significant increase in latency when the file size changed from 145KB to 294KB.

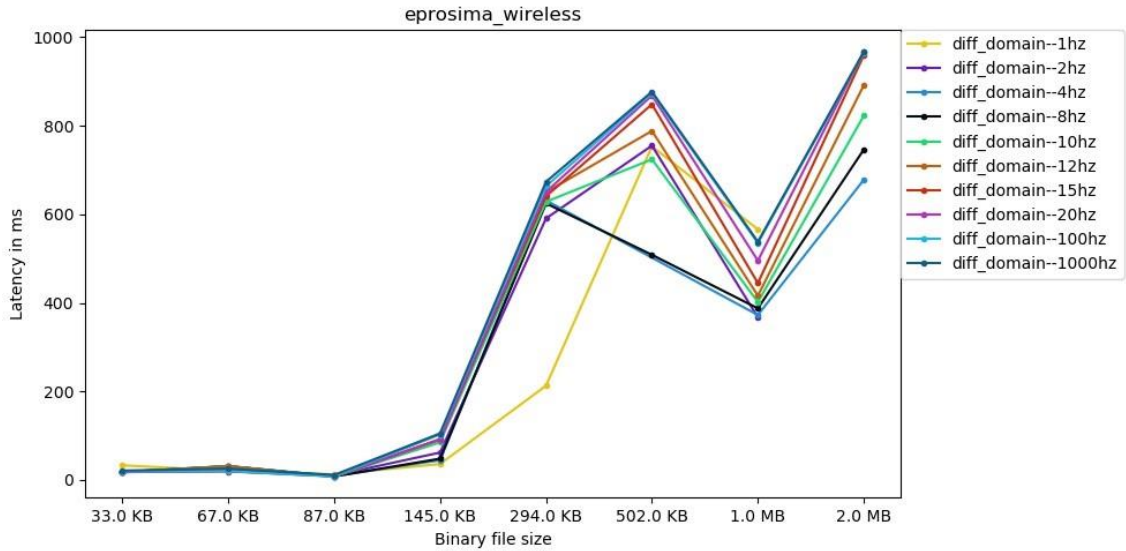


Figure 4-9. Publisher frequency impact on the latency using eProxima DDS different domain wireless communication between Pi4.

Laptop Computer: When laptops are connected wirelessly within the same domain and using eProxima DDS, we observed different latency characteristics for file sizes of 145KB, 1MB, 2MB, and 4MB across all publisher frequencies. It is interesting to note that for lower frequencies like 1Hz and 2Hz, we noticed higher latency for 145KB files, and similar behavior is observed for 1MB files at 4Hz, 8Hz, 10Hz, and 12Hz frequencies. Additionally, the 4MB file size had the poorest performance at 20Hz and 100Hz frequencies.

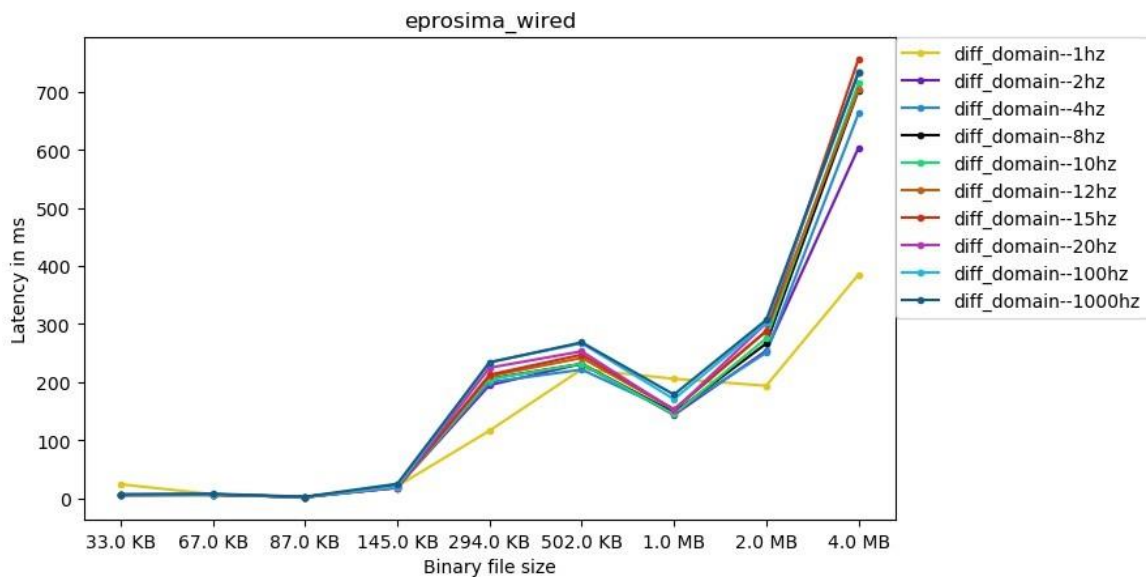


Figure 4-10. Publisher frequency impact on the latency using eProxima DDS different domain wired communication between Laptop.

Based on our observations, we have discovered that when using eProsima DDS to communicate between laptop nodes in different domains through a wired connection, there is minimal latency difference for most frequencies (except for 1Hz) when dealing with file sizes up to 87KB. However, we have noticed an increase in latency for files up to 502KB, followed by a decrease for files of 1MB in Figure 4-10. It is worth noting that we have seen a significant rise in latency when the file size changes from 2MB to 4MB.

4.3.1.3 RTI Connex DDS

RTI is one of the leading providers of software frameworks for smart machines and real-world systems, and their flagship product, RTI Connex, is the ultimate solution for seamless and secure information exchange between multiple applications, ensuring a unified and efficient system. Connex is built on the foundation of the DDS standard, providing a reliable and low-latency availability that is absolutely essential for real-time systems to operate without any interruptions. Furthermore, with Connex, development, integration, and maintenance costs significantly decrease. The data awareness capability of the framework boosts the data flow optimization to find the appropriate connection and desired performance.

Connex DDS provides the data filtering capability, optimizing bandwidth usage in a complex system and ensuring that only relevant data is delivered to each application. Connex's peer-to-peer architecture eliminates the need for brokers or servers, allowing for efficient message transfer between publishers and subscribers with low overhead. The platform handles the broker's routing, discovery and naming functionality in a distributed, lightweight, and reliable manner. Thus there is no requirement for specialized hardware or server software.

Raspberry Pi3: When transferring binary data wirelessly between Pi3 same domain nodes using RTI Connex DDS, we noticed that the latency pattern is similar to that of Eclipse Cyclone DDS. We found that up to a file size of 145KB, the different frequencies do not affect latency. However, when transferring a 502KB file, we observed higher latency compared to a 1MB file. Additionally, we found that lower frequencies, such as 1Hz or 2Hz, produced higher latency for larger files, while medium frequencies like 8Hz and 10Hz performed better than higher frequencies, which was different from other DDS implementations.

While transferring binary data between wirelessly connected Pi3 different domain nodes using RTI Connex DDS, in Figure 4-11 we observed that different frequencies for smaller file sizes similarly impact the latency. Here also 502KB file yields higher latency than the 1MB file.

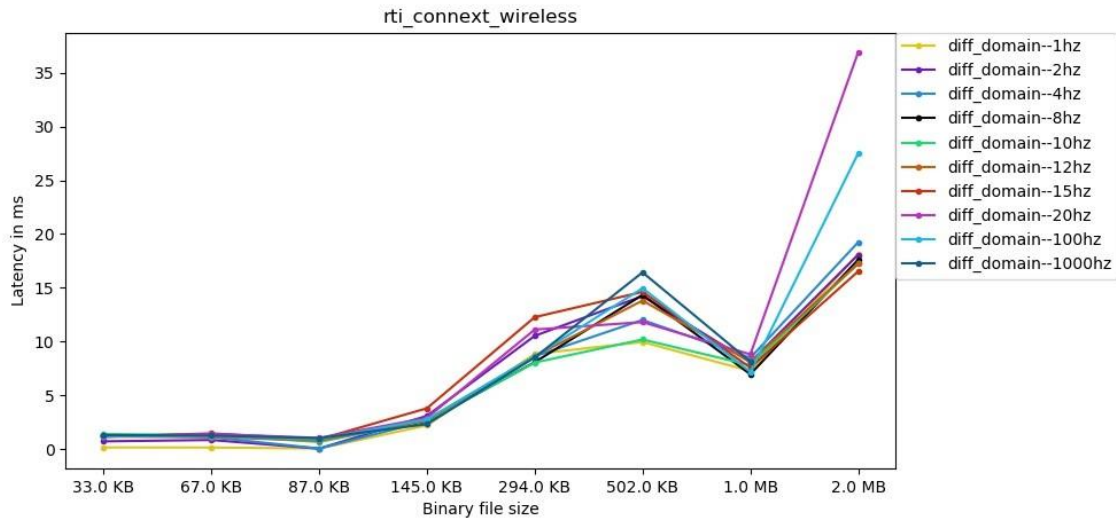


Figure 4-11. Publisher frequency impact on the latency using RTI Connex DDS different domain wireless communication between Pi3.

We have observed that when using RTI Connex DDS with a wired connection, same domain Pi3 nodes, there is little difference in latency between 502KB and 1MB file sizes at lower frequencies of 1Hz and 2Hz. However, when communicating between wired nodes in different domains, we have found that the latency is higher for 502KB files compared to 1MB binary files in Figure 4-12, as seen in the case of Eclipse Cyclone DDS. It is also important to note that there is a significant increase in latency when the file size increases from 145KB to 294KB for all of the publisher frequencies.

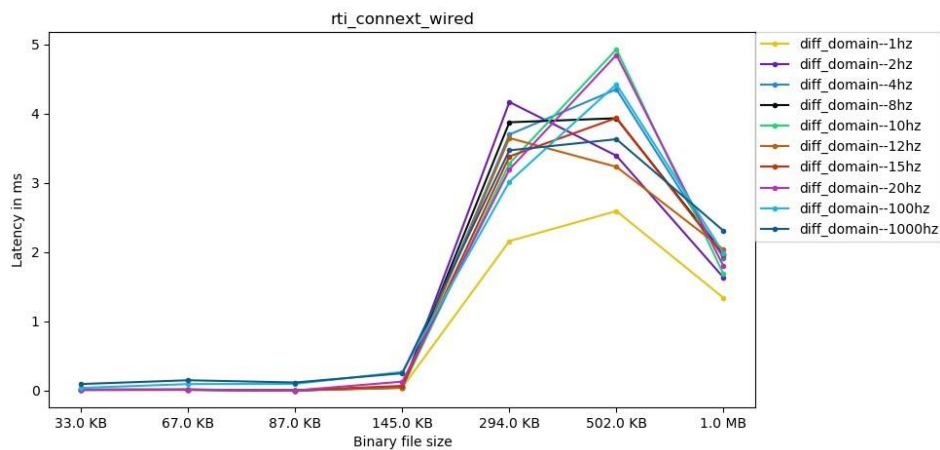


Figure 4-12. Publisher frequency impact on the latency using RTI Connex DDS different domain wired communication between Pi3

Raspberry Pi4: When using RTI Connex DDS to transfer binary data wirelessly between nodes in the same domain, we have discovered that for files larger than 145KB, the frequency used has a direct effect on latency. Lower frequencies are more efficient for transferring larger files. However, when transferring data wirelessly between different domain Pi4 nodes using RTI Connex DDS, we have found that all frequencies except for 20Hz and 15Hz have the same impact on latency for different file sizes in Figure 4-13. This is different from the results obtained when transferring data wirelessly between the same domain Raspberry Pi3 and laptop nodes.

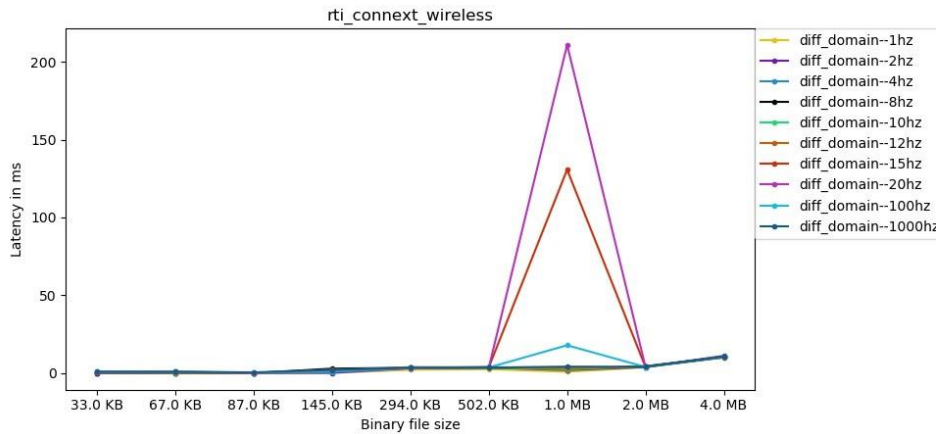


Figure 4-13. Publisher frequency impact on the latency using RTI Connex DDS different domain wireless communication between Pi4

When comparing wired connected Pi4 nodes on the same domain and using RTI Connex DDS, the latency pattern remains consistent for file sizes up to 145KB. However, it is interesting to note that a 294KB file has higher latency compared to a 502KB and 1MB file size for most frequencies. Additionally, lower frequencies perform better for larger files in this experiment. In our observation of communication between Pi4 wired nodes in different domains, we noticed that sending a 294KB file results in higher latency compared to a 502KB file. However, when it comes to higher frequencies such as 12Hz, 15Hz, 20Hz, and 100Hz, a 4MB file has lower latency than a 2MB file. Additionally, a lower frequency also shows better performance for larger file sizes in this experiment.

Laptop Computer: Laptop nodes within the same domain that are wirelessly connected and using RTI Connex DDS exhibit unexpected behavior when transferring small files. Interestingly, frequencies like 8Hz and 10Hz experience higher latency when transferring a 33kb file than a 4MB file in Figure 4-14. However, when transferring a 145KB file, all frequencies (except for 4Hz) result in higher latency. The latency improves and behaves consistently when transferring larger files beyond 145KB.

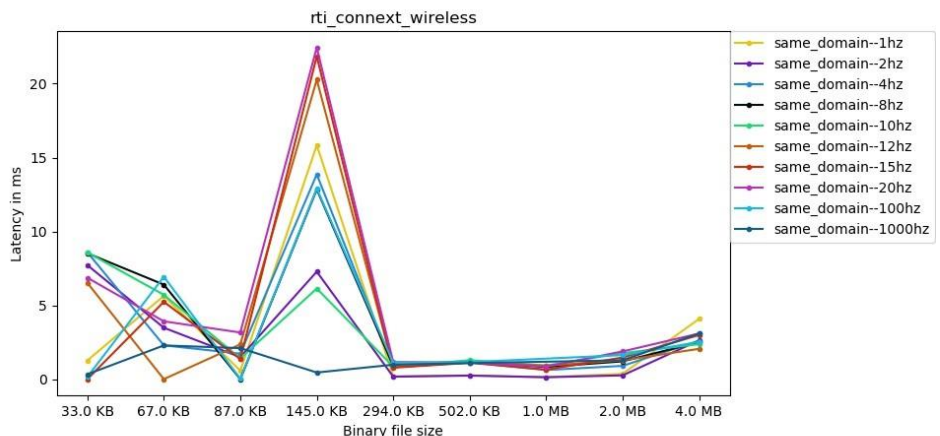


Figure 4-14. Publisher frequency impact on the latency using RTI Connex DDS same domain wireless communication between Laptop

When communicating between laptop nodes that are wirelessly connected and reside in different domains, we have noticed that most frequencies have a similar impact on different file sizes. As file size increases, latency also increases until it reaches a 502KB file size, after which it drops slightly for 1MB files figure. Interestingly, we found that the latency for 294KB and 2MB files is almost the same. Lower frequencies like 1Hz, 2Hz, and 4Hz perform better for larger file sizes.

Observations and Analyses:

For most of the experiments, we have noticed the impact of publisher frequency on latency. For wirelessly connected Raspberry Pi3 nodes while using eProsima DDS, we observed that nearly all the publisher frequencies have the same resultant latency for all file sizes; this phenomenon was quite unexpected. For all the DDS implementations, regardless of publisher frequencies and domain communication, most of the use cases 502KB binary files yield more latency than 1MB binary files. With RTI Connex DDS wired Pi4 nodes for higher publisher frequencies, the latency of 4MB files is smaller than the 2MB files. Up to file size 145KB for the binary files eProsima DDS, Eclipse Cyclone DDS and some instances of RTI Connex DDS have shown similar latency characteristics regardless of publisher frequencies. However, wirelessly connected laptop nodes using RTI Connex DDS have different latency characteristics for different publisher frequencies for smaller file sizes up to 145KB. A sudden rise in latency has been observed for all the DDS implementations while the binary file size changes from 145KB to 294KB. While transferring larger files, in most cases, lower frequencies have better latency. However, for wired connected Pi4 nodes using eProsima DDS, we observed that 1Hz and 2Hz publisher frequencies performed worst for the large files.

Varying results were exhibited by different DDS systems when comparing communication performance across different domains and within the same domain for various file types and sizes. For wirelessly connected Pi3 nodes, Eclipse Cyclone DDS with smaller binary files and lower publisher frequencies, the same domain communication performs better, but the situation changes as the file size reaches 294KB; different domain communication behaves better. Different domain communication performs better when the binary file exchange happens between the wirelessly connected laptop nodes using Eclipse Cyclone DDS for all file sizes and publisher frequencies. Nevertheless, with wired connectivity, the same domain communication for all physical devices using Eclipse Cyclone DDS performs better than different domain communication. When the different physical nodes use eProsima DDS to exchange binary data regardless of the binary file sizes, publisher frequencies, and communication medium, the same domain communication consistently outperforms the different domain communication.

When wirelessly connected Pi3 nodes use RTI Connex DDS for the smaller binary file, both the same domain and different domain show similar latency characteristics, but when the file size becomes 145KB different domain outperforms the same domain communication for all publisher frequencies. On the wirelessly connected Pi4 node for larger files with lower frequencies 2Hz same domain communication yields lower latency, but with 8Hz frequency for the same file sizes, different domain communication outperforms the same domain communication. Different domain communication works better for all publisher frequencies and file sizes for the wired connected laptop nodes using RTI Connex DDS.

From the above discussion, we observe that there is no rule of thumb for vendor-specific DDS implementation performance. The performance of DDS implementation depends on the

exchanged file type, communication media type, and file size. Different domain communication performed better for the larger binary file exchange in some cases.

4.3.2 P2P-based Discovery and Federation

This work focuses on storing and querying RDF data obtained from distributed sources across a network of dynamic swarm intelligence nodes that are represented by a network of edge devices in a fully distributed fashion. By employing a solution to integrating both the structured P2P system, P-Grid [Aberer01], and an edge-based RDF storage, RDF4Led [Anh18], this new design enables maintaining RDF data storage among distributed swarm nodes and query processing across swarm nodes while scaling with increasing data and network size.

4.3.2.1 System Architecture and Implementation

The implementation integrates the RDF4Led engine and P-Grid system to create a distributed RDF store for a P2P network of lightweight edge devices. It utilizes the flash-friendly RDF storage of RDF4Led and the P-Grid virtual binary search tree to efficiently manage and query RDF data on each node in the network. The Figure 4-15 below illustrates the architecture overview of integrating the RDF4Led and P-Grid components on a single swarm intelligence node.

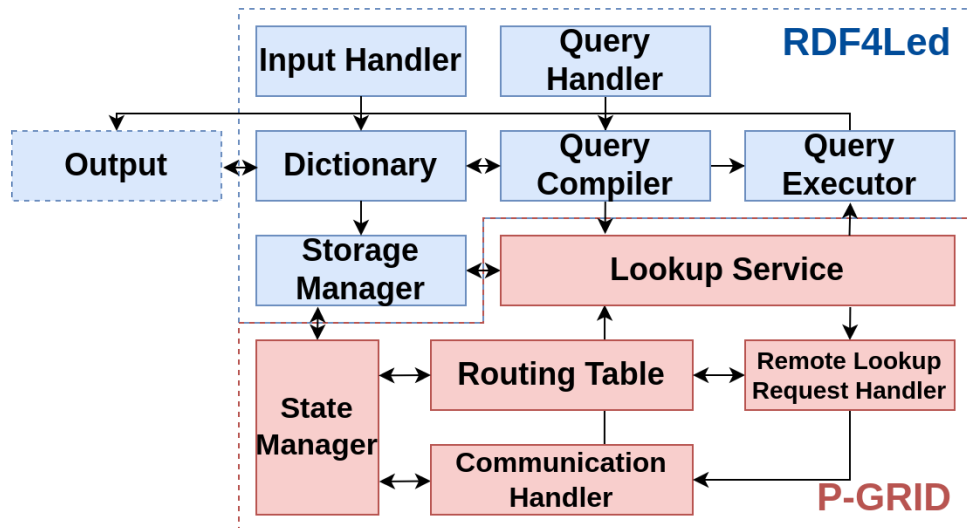


Figure 4-15. Architecture Overview of P-Grid and RDF4Led's Integration

The critical components of the design to be extended are the RDF storage and SPARQL query processor of RDF4Led, and the State Management and Lookup Service of P-Grid. Here, the blue part represents the original architecture of RDF4Led consisting of an Input Handler that is tied to a Dictionary to translate between string-based RDF resources and encoded identifiers. Dictionary adopts a hash function to create a fixed-length integer deterministically as a representation of an original string of arbitrary length. Because of its natural behavior, the hash function is suitable for key-value structures. The encoded RDF triples are indexed with three index layouts (SPO, POS, OSP) and are stored with a Storage Manager that employs a two-layer index for each layout as presented in Rdf4led.

SPARQL queries are registered on the system via a Query Handler and are compiled with a Query Compiler. For compiling a SPARQL query, the Dictionary will involve converting RDF nodes in basic graph patterns to encoded identifiers. A Query Executor is implemented to execute the query plans computed by the Query Compiler and to produce the output results. The Output Handler returns the original format of RDF resources for these output results from the Query Executor.

The red part encompasses essential functions adopted from P-Grid. The State Manager from P-Grid serves as a controller for a peer, facilitating state transitions based on given inputs. It includes primary states, such as the bootstrapping phase, exchange phase, replicating phase, and running phase.

The bootstrapping phase initiates when a swarm node/peer joins the P2P network, aiming to discover and familiarize itself with other participants. Subsequently, during the exchange phase, existing peers in the P-Grid overlay structure undergo stabilization, but data distribution might remain imbalanced. To address this, the exchange phase reorganizes data items among RDF peers.

A static approach with a global replication factor of two ensures that each data item has two replicas in the P2P network. During the exchange phase, only data blocks are replicated, with each replica recording the origin peer containing the actual RDF triples within the block. Origin peers halt initiating replicating requests until their data blocks meet the global replication requirement.

Once the exchange phase is complete, the running phase commences, making a peer ready to work. Peers in this phase can both initiate query requests and respond to queries from other peers in the P-Grid network.

Throughout each phase, the State Manager communicates through a Communication Handler, facilitating message exchange. The Lookup Service triggers lookup requests to the Remote Lookup Request Handler, which forwards requests to other peers. The Routing Table aids the State Manager and the Remote Lookup Request Handler in identifying the peers to communicate with.

With this architecture, each peer in the network has an RDF4Led Storage Manager responsible for storing and maintaining the RDF data locally. The Storage Manager handles data insertion or deletion and resolves query requests. If new data needs insertion or updating, the Dictionary will first encode the string into an identifier to accelerate the search and save the memory space in the Storage Manager. The design of the flash-aware storage layout and indexing scheme of a single RDF4Led machine are in use as they cater to the need for a suitable storage method for lightweight edge devices. Hence, the Storage Manager contains a buffer layer and a physical RDF storage layer. The data in the physical layer is organized as data blocks; the buffer layer is the index of each data block in the physical layer. In our system, the indexes of the data blocks are published to the State Manager. Using the peer information from the Routing Table and based on the indexed key, the State Manager will decide which data block should be replicated or exchanged to which peer to maintain the load balance for the network.

To retrieve RDF triples from the Physical Layer, the Storage Manager initially searches the Buffer Layer to identify the indexes of the data blocks potentially containing the desired results. Subsequently, the Storage Manager accesses the encoded values from the Physical Storage Layer, utilizing the key value of each data block. This retrieval allows the Storage Manager to further decompose the encoded value into multiple tuples, facilitating subsequent result trimming.

After compiling a SPARQL query, the Query Compiler computes an optimized query plan. Each triple query request of the query plan is resolved by the Lookup Service, which will search in the local storage of a peer or forward the request to remote peers. The search mechanism in the P2P system is indicated by Routing Table, which is essential for a structured P2P overlay, as it

holds the information of other peers. The Routing Table ensures a triple query request is answered by a particular peer if the requested data exists in the overlay. When matched triples are found in a peer, the result sets are forwarded back to the Query Executor as a final or intermediate result. The result generated by the Query Executor would be translated by the Dictionary back to the original format of the triples as the output.

4.3.2.2 Preliminary Experimental Findings

This system is developed in Java and reuses as much of the source code from RDF4Led and P-Grid as possible. We also re-implemented some parts using updated technologies. For instance, we recycled the dictionary module from RDF4Led and the bootstrapping mechanism from P-Grid. The Java WebSocket implementation in the initial version of P-Grid was replaced by gRPC to improve the system's ability to handle asynchronous message passing.

We conducted our experiments using a cluster of 4 to 16 Raspberry Pi 4 (Pi4) devices, which serve as lightweight and cost-effective edge devices for the IoT. Each device is equipped with quad-core processors clocked at 1.5GHz, 8GB of RAM, and an onboard LAN connection with a speed of 1Gbps. Peers are considered directly interconnected with every other peer in the experiments.

For our experiments, we utilize the ISD (Integrated Surface Dataset) [ISD24], a notable weather dataset comprising weather observations collected from 20 thousand weather stations worldwide since 1901. This dataset encompasses various measurements, including temperature, wind speed, wind angle, and more. Moreover, each observation is accompanied by timestamps indicating when these measurements were recorded.

```
PREFIX sosa:<http://www.w3.org/ns/sosa/>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?observation
WHERE { ?observation rdf:type sosa:Observation. %TP1 }
```

Figure 4-16. Atomic Triple Pattern- Listing all observations

Experiment 1: QET of a Single Atomic Triple Pattern

To initiate the study of our system's behavior when responding to a SPARQL query, we measured the QET (Query Execution Time) of a SPARQL query containing a single atomic triple pattern, as depicted in Figure 4-16. Given that this query doesn't entail any join operations, this experiment aims to offer an analysis of how message passing within a P2P network influences the QET of such a P2P RDF engine.

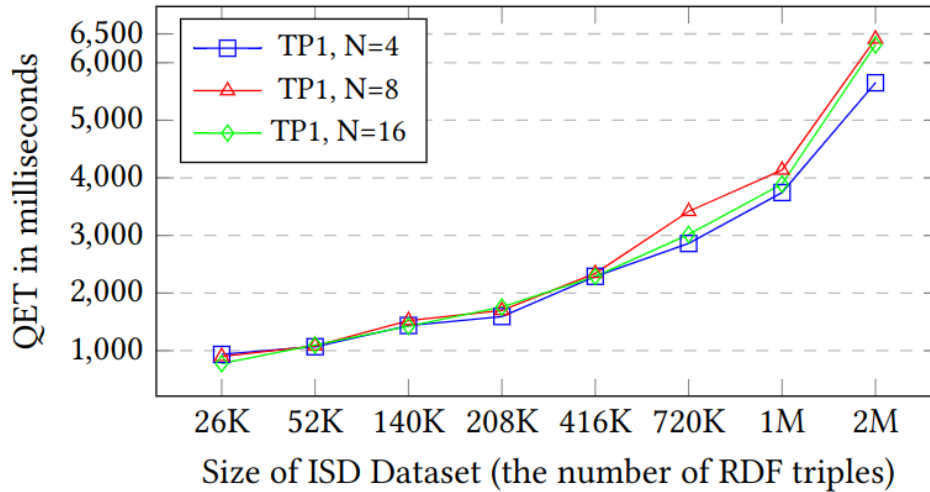


Figure 4-17. QET of Atomic Triple Pattern TP1 Using ISD Dataset. N is the number of peers in the system

Figure 5.3.2.2

It is essential to note the measurements we obtained here regarding the IO delay within our setup. Through a microbenchmark of network IO, we determined that the act of sending 1000 messages, each of 1KB in size, consumes approximately 1147 milliseconds. It's worth noting that the delay in local storage IO is notably minor in comparison, rendering it inconsequential when compared to the time taken for communication.

As mentioned in the previous section, the number of triples is divided approximately equally among the peers involved, indicating that the network achieved a balanced key distribution after multiple data exchange phases during P-Grid construction. With N participating nodes, the query initiator required at most $\log(N)$ hops to locate the results.

Under these data setup conditions, we varied the size of the ISD dataset to 26K, 52K, 140K, 208K, 416K, 720K, 1M, and 2M, as shown on the x-axis. Consequently, this led to varying numbers of RDF triples being returned for the atomic triple pattern: 2K, 4K, 10K, 16K, 31K, 54K, 75K, and 153K. It is worth noting that in this scenario, the size of the result set accounted for nearly 8% of the total dataset. The number provided is significantly larger than the actual result size typically returned from a SPARQL query, which often falls below 1% or even 0.1%. We measured the QET by recording the time from the initiation of a request until the initiator received all matching results from the answering nodes. The test results for query execution time when responding to a single atomic triple pattern on different data scales in our setup are presented in the Figure 4-17.

As shown in the Figure 4-18, our system experiences delays in searching and retrieving data, ranging from 1 to 6.5 seconds, across datasets comprising 26K to 2M triples. Throughout the querying process, the communication cost encompasses several factors, including the hops required to locate answering peers, the expense incurred as answering peers transmit messages containing possible block entries, the outlay for the query initiator to request matching RDF triples for each block entry received from answering peers, and the cost for answering peers to send messages containing matching RDF triples.

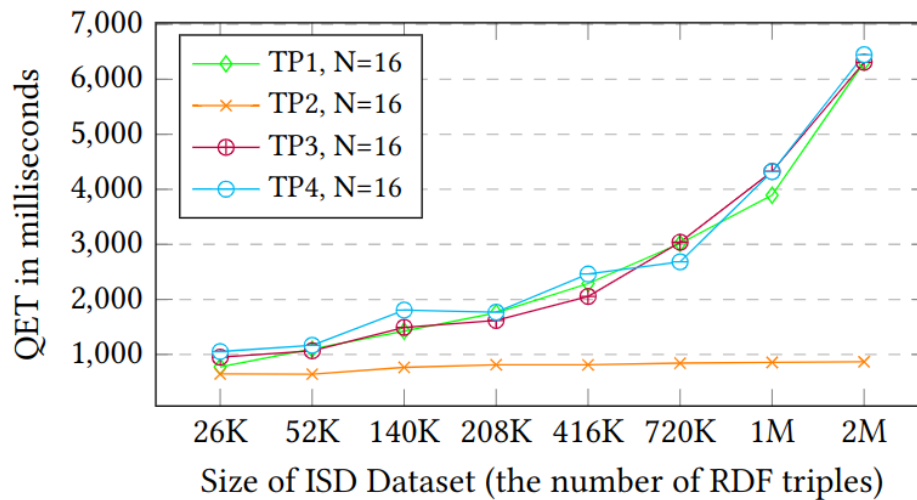


Figure 4-18. QET of Atomic Triple Patterns TP1~4 Using ISD Dataset on 16 Pi4s

Furthermore, the results shown in the Figure 4-18 highlight that QET is significantly influenced by the number of matching RDF triples returned. Increasing the dataset size leads to a considerable delay increase. In this context, the difference in QET across various network sizes is not very significant. Increasing the number of involved nodes results in slight delays. This is primarily because, when considering datasets of the same size, the number of matching RDF triples remains constant, with only one or two hops added during the searching phase.

Experiment 2: QET of Complex Join Query Patterns

```

PREFIX sosa:<http://www.w3.org/ns/sosa/>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT ?obs ?featureOfInterest ?obsProperty
WHERE {
%sensor% sosa:madeObservation ?obs.           %TP2
?obs sosa:hasFeatureOfInterest ?featureOfInterest. %TP3
?obs sosa:observedProperty ?obsProperty.      %TP4}

```

Figure 4-19. Join Query Pattern containing 3 atomic TPs -Listing information of all observations made by a sensor

To gain further clarity on the impact of message passing quantity, we repeated the experiment using various triple patterns (TPs). To avoid redundancy, we present results from our 16-node network. Figure 4-18 depicts the test outcomes utilizing a triple query pattern from the 2nd SPARQL query, employed in our second experiment. Given the similarity in the number of matched triples between TP3 and TP4 in the query shown in Figure 4-19, and TP1 in the query presented in Listing 1, the delays are almost the same.

For TP2, we fixed the subject %sensor% to a specific sensor IRI, resulting in a fixed number of matched triples and returned results, even as the data scale increased. The QET remains consistent despite the growth in data size. Our system achieved the capability to return around four thousand results within less than a second in the context of a 16-node system.

Using an ISD dataset of 26K triples, and a cluster of 16 Pi4s, the QET for the join query, as illustrated in Listing 2, was found to be 11.15s. To extrapolate the execution time of join queries with uniform data distribution across various dataset sizes and network scales, we are prompted to employ synthetic data to execute an analogous join query.

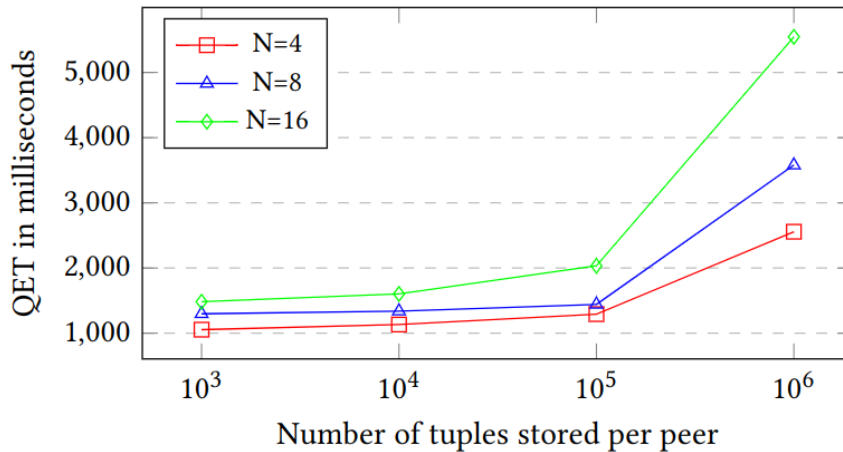


Figure 4-20. QET of Multiple Join Operations with Uniform Data Distribution

Figure 4-20 presents the test results. As anticipated, the query execution time increases with the number of answering nodes and the storage size of each peer. The figure illustrates that there is a direct correlation between the execution time of the join query and the number of peers participating in the query. This suggests that the more peers are involved in the join query, the longer it takes to complete the query due to the increased communication overhead. Furthermore, a significant rise in execution time is observed when the number of tuples per peer reaches 1M. However, when the number of tuples per peer remains below 10^5 the execution time shows little variation. This phenomenon may be attributed to the longer search time required for each answering peer in its local storage with a substantially larger dataset, resulting in an increased number of messages in transit.

4.4 COMPONENT DESIGN

The detailed design of the Orchestration & Optimization system, as depicted in Figure 4-21, is a pivotal component slated for development in task T5.3 of the project. This system is designed to enable adaptive coordination and optimization, crucial for handling the dynamic and evolving requirements of a SmartEdge swarm. To fulfill this objective, T5.3 will introduce two key subsystems: the Orchestrator and the Optimizer, as introduced in Section 5.1. The Orchestrator is tasked with orchestrating tasks from application specifications, while the Optimizer focuses on analyzing data and processes to enhance performance and efficiency. Both of these subsystems will be tightly integrated with the SmartEdge runtime, a component developed in task T5.4. The SmartEdge runtime will execute plans generated by the Orchestrator, indicating its role in translating orchestration strategies into actionable tasks. This holistic approach ensures seamless coordination and optimization within the system, aligning with the project's overarching goals of efficiency and adaptability.

A task, defined as a semantic program, can be assigned to a smart node through the Task Register of the orchestration subsystem. This registration process provides the mechanism for smart nodes to receive and execute specific tasks within the SmartEdge swarm architecture. There are two primary methods for assigning a program: it can be directly sent from the

toolchain compiler or transferred from the Orchestrator of another SmartEdge smart node. In the first method, the task is transmitted directly from the toolchain compiler, which is a part of the development environment used to create programs for the smart nodes. In the second method, the task is transferred from the Orchestrator of another smart node within the swarm, enabling distributed task assignment and execution. Once a task is received, it undergoes validation by a Task Validator. This validation step ensures that the task is compatible and suitable for execution within the swarm environment. Specifically, the Task Validator checks whether the task is responsible for the swarm, meaning it does not require the node to serve other swarms or conflicting responsibilities. This validation process helps maintain the integrity and efficiency of the SmartEdge swarm by ensuring that tasks are appropriately allocated and executed across the nodes.

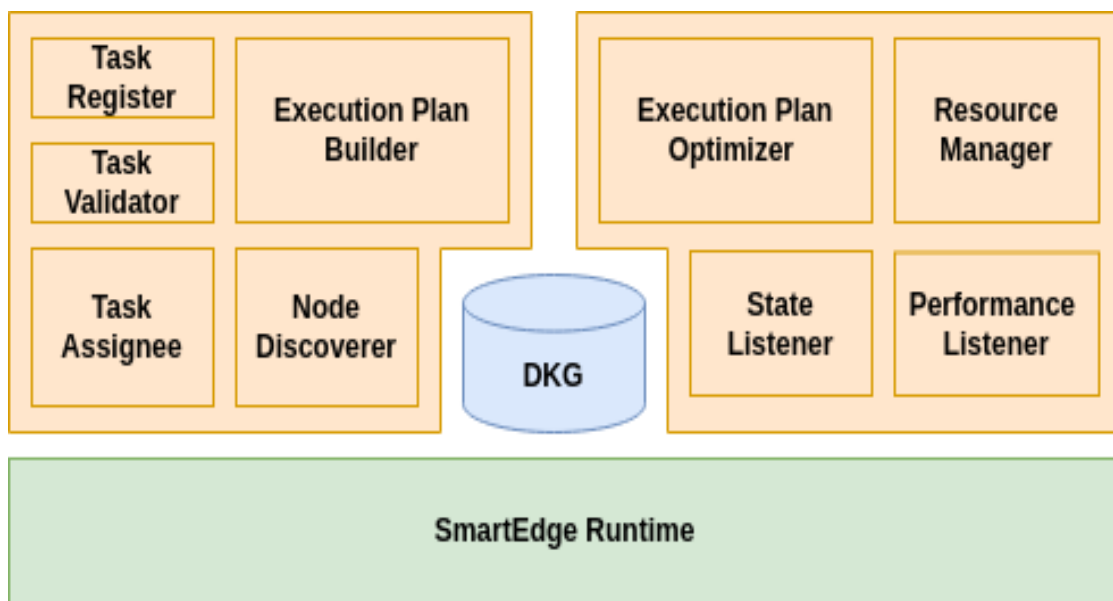


Figure 4-21. System Architecture of the Coordinator & Optimizer

The semantic program is then sent to the Execution Plan Builder which constructs an execution plan to outline how the program will be executed within the SmartEdge swarm environment. This execution plan is a strategic roadmap that determines the sequence of actions needed to fulfill the program's objectives. To ensure optimal performance and resource utilization, the execution plan undergoes optimization through the subsystem, Execution Plan Optimizer. This optimization process fine-tunes the plan by analyzing factors such as computational complexity, data dependencies, and network constraints. When a task is offloaded to a special node with hardware acceleration, the optimizer calls the runtime optimizer of T5.2 (see above) to optimize the hardware acceleration part of the operation.

The optimized execution plan specifies the tasks that can be executed locally on the current smart node, as well as those that may need to be delegated to other nodes within the swarm. In some cases, certain tasks may need to be federated or distributed across multiple smart nodes to achieve efficient execution. This decision is based on factors such as task complexity, available resources, and network bandwidth. The execution plan itself consists of a sequence of SmartEdge processing primitives (cf Section 5.4.3 which are fundamental processing operations that the SmartEdge Runtime can execute. By organizing the program's execution into a series of primitives, the system ensures that each task is broken down into manageable steps that can be efficiently executed by the SmartEdge Runtime across the swarm of smart nodes.

The Task Assignee plays a critical role in orchestrating swarm operations by facilitating task assignment and node recruitment within the distributed system. Initially, it undertakes the task of swarm formation, leveraging its authority to allocate tasks to swarm participants. This involves the intricate process of task distribution, commencing with the assignment of primitives to smart nodes. Upon analysis, if the primitives are deemed suitable for local execution, the Task Assignee seamlessly dispatches the primitive descriptions to the SmartEdge Runtime. These descriptions encapsulate optimized parameter configurations generated by the Optimizer subsystem, ensuring efficient execution of primitives on the designated smart nodes. However, in scenarios where segments of the execution plan necessitate delegation to other nodes, the Task Assignee employs a sophisticated strategy. It reconstitutes the execution plan into a semantic program and initiates transmission to the relevant smart node(s), ensuring coherent task execution across the swarm. While the process of converting back to a semantic program may introduce unnecessary steps, it provides a generalized description of the task, allowing for further optimization by other nodes based on their contextual understanding. Furthermore, the Task Assignee actively engages in the recruitment of SmartEdge nodes, spanning diverse devices such as edge systems, vehicles, sensors, and robots, thereby bolstering the swarm's capabilities. This recruitment process mirrors the intricacies of the matchmaking mechanism introduced in WP2, meticulously identifying and soliciting participation from nodes best suited to serve the swarm's objectives. Integrated seamlessly with a Node Discoverer, the Task Assignee harnesses the power of a Dynamic Knowledge Graph to catalog and maintain metadata pertaining to discoverable nodes and swarm participants. Leveraging this repository, the Node Discoverer formulates SPARQL queries tailored to the specific requirements of task execution or primitive operations, enabling precise node discovery and recruitment.

The Dynamic Knowledge Graph can be implemented as an RDF store containing metadata pertaining to discovered nodes within the network. This metadata includes specifications of the devices and their real-time state, including factors such as location or resource availability. The Dynamic Knowledge Graph can either be centralized on a smart node, serving as a hub for swarm coordination, or distributed across multiple smart nodes. In the centralized model, a single smart node assumes the responsibility of maintaining and updating the Dynamic Knowledge Graph. Alternatively, in the distributed model, fragments of the knowledge graph are distributed across multiple smart nodes, allowing for decentralized management and redundancy. Using RDF to annotate participants' metadata enables the discovery process through querying the metadata with SPARQL.

In the Optimizer subsystem, the State Listener and Performance Listener play pivotal roles in updating the state of the smart edge nodes and monitoring the current performance of primitives. These listeners continuously gather metrics, including those from the resource manager, which serve as inputs to the Execution Plan Optimizer for optimizing the execution plan. This updating process occurs dynamically at runtime, ensuring that the optimization strategies are informed by real-time data. The Execution Plan Optimizer devises strategies to optimize the execution plan based on the runtime metrics it receives.

4.5 SEMANTIC-BASED DISCOVERY AND FORMATION OF SWARM

As mentioned previously, the formation of a swarm mandates that each SmartEdge node exposes itself along with its semantic description. Even for SmartEdge nodes lacking advanced SmartEdge services and protocols, it is indispensable to register their specifications within the knowledge graph. This ensures comprehensive visibility and accessibility of all nodes' capabilities for seamless coordination and collaboration within the swarm.

The registration process can be conducted manually during the design phase, wherein specifications are meticulously documented and subsequently uploaded to a cloud repository. Alternatively, specifications can be dynamically added at runtime through the utilization of smart edge message middleware, facilitating real-time updates to the knowledge graph as nodes join or modify their configurations.

```
1 {
2   "@context": "https://www.w3.org/2022/wot/td/v1.1",
3   "title": "Camera number at Junction 270",
4   "id": "urn:uuid:9489991a-7622-45b6-8437-f859835d4",
5   "geo:lat": "60.16453",
6   "geo:long": "24.912846",
7   "events": {
8     "traffic_images": {
9       "forms":
10      [{
11        "href": "RTSP://helsinki.fi/camera/270/",
12        "contentType": "video/mp4"
13      ]}]
14   }
15 }
```

Figure 4-22. JSON-LD snapshot of semantic description of the camera located at Junction 270 in Helsinki

Figure 4-22 presents a detailed depiction of the semantic metadata associated with the camera installed at Junction 270 in Helsinki, designated as the deployment site for UC2. For a comprehensive understanding of the traffic network layout and camera deployments, readers are directed to section 5.4.1.1. The data depicted in the figure is structured in JSON-LD format, adhering meticulously to the Web of Things Thing Description (WOT-TD) ontology specified in WP2. This structured representation encapsulates information about the camera, facilitating its seamless integration and operation within the SmartEdge ecosystem. Lines 3 and 4 of the metadata furnish essential particulars such as the camera's title and its unique identifier within the SmartEdge ecosystem. This distinct identifier ensures the camera's unambiguous identification and enables seamless communication and interaction within the ecosystem. Additional properties, such as the camera's precise geographical coordinates (longitude and latitude), can be included to augment its contextual awareness. Lines 11 and 12 delineate the precise mechanism for accessing the video stream originating from the camera.

The advanced functionalities of SmartEdge Runtime, elaborated upon in Section 5.4.2.2, empower all smart nodes to conduct hardware specification scans during the bootstrapping phase. Additionally, the SmartEdge Runtime furnishes information regarding available service hosts on each node. Leveraging the hardware specifications, the toolchain can ascertain the capabilities of each smart node. These specifications are meticulously documented using the Resource Description Framework (RDF) and subsequently stored in the local dynamic knowledge graph. This robust repository forms the foundation for efficient resource allocation and seamless coordination among nodes within the swarm.

An autonomous smart node seamlessly integrates into an existing network of peers by autonomously introducing itself to other nodes within the network. This is achieved by transmitting its semantic description to neighboring smart nodes, enabling them to identify and establish connections with the newcomer. The description encompasses crucial details such as endpoints for the node's available services, thereby facilitating streamlined communication and collaboration among nodes within the swarm.

Figure 4 depicts an exemplary subscription message originating from a smart node, crucial for conveying detailed information about its capabilities and resources within the network architecture, thereby facilitating efficient task assignment and coordination. The subscription message delineates the endpoint for task assignment, from Line 7 to Line 17, providing a precise reference for directing tasks within the network architecture. Subsequently, from Line 18 to Line 33, comprehensive insights into the device's hardware configuration, including CPU, RAM, and GPU specifications, are meticulously provided. This data empowers decision-makers to allocate tasks optimally based on resource requirements and constraints. Moreover, from Line 35 to Line 40, specific device capabilities or skills are highlighted, such as the capability for object detection from images. This information enables nodes to allocate tasks aligned with the device's specialized functionalities effectively.

Figure 4. Json-LD snapshot of semantic description of a smart node

4.5.1 RDFizing P4-based network information into Dynamic Knowledge Graph

This section outlines the operational framework wherein swarm nodes leverage P4-based network information for both discovery and querying functionalities. The integration of the P4-runtime API facilitates seamless communication channels between processes or controllers, acting as gRPC clients, and the data plane elements of swarm nodes, operating as gRPC servers.

A prime illustration of this operational paradigm is depicted in Figure 4-23, wherein a P4 controller interacts with the P4 target via P4Runtime protocols. Within this context, the pivotal role of the P4RDFizer, as shown in Figure 4-23, becomes apparent. Its core function involves extracting crucial packet header information and encapsulating it into RDF objects. These RDF data is then dynamically stored within knowledge graphs during runtime.

The crux of this process revolves around the transformation of P4-based data into RDF-compliant structures. This facilitates the execution of intricate queries and analyses on the network's behavior and characteristics. Additionally, the utilization of dynamic knowledge graphs ensures the accessibility and updateability of information, thus accommodating the dynamic nature of modern network environments.

Moreover, the integration of packet I/O mechanisms enables the seamless streaming of packets from the data plane to the control plane via dedicated CPU ports. This facilitates further inspection and analysis of network traffic, empowering administrators with deeper insights into network performance and security.

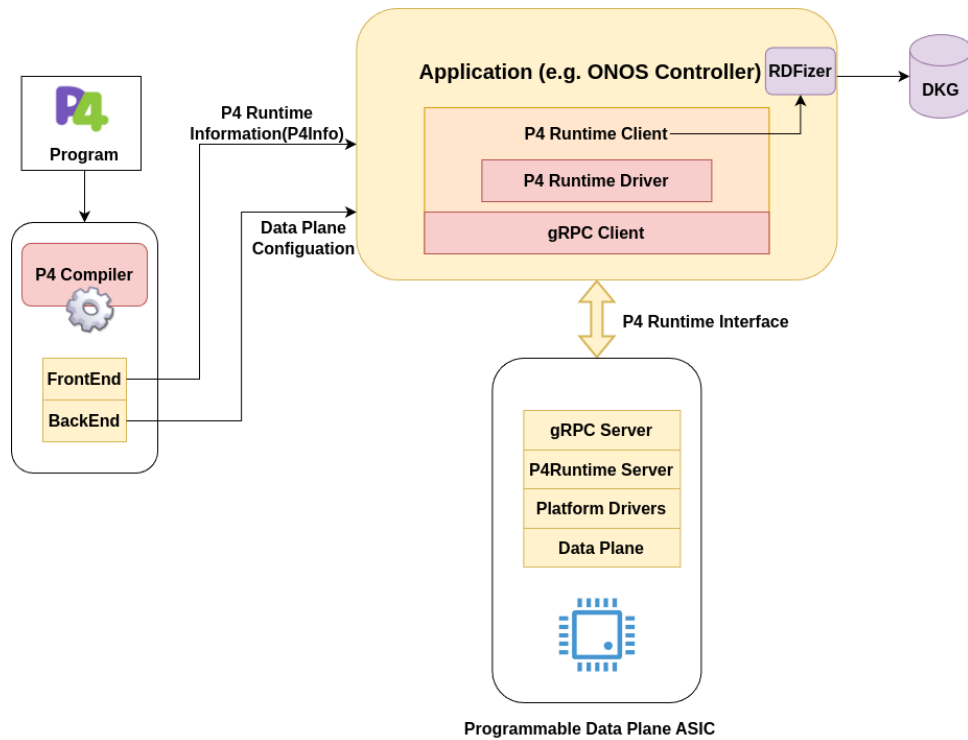


Figure 4-23. Integration of Dynamic Knowledge Graph and P4 Runtime for Packet-Level Metadata Collection

Various kinds of information obtained from the packets flowing through the P4 switch can be stored within the dynamic knowledge graph (KG) inside the P4 controller node, such as ONOS [Onos24]. For example, in Figure 4-24, two types of metadata information are annotated in RDF data format: network-specific and swarm node-specific. As advocated in Deliverable D4.1, an innovative design for in-band telemetry can be activated for all swarm nodes. Besides traditional network-specific metadata such as switch ID and hop latency, the SmartEdge packet header includes innovative augmented metadata fields such as CPU load and device localization information. These rich metadata regarding the network state are cloned to the controller via packet I/O operations for further inspection and are extracted as RDF objects inside the P4 ONOS controller. They are then stored as RDF triples for the dynamic knowledge graphs at the controller node. The controller node can be single or multiple; thus, the dynamic knowledge graphs can be centralized or distributed. Taking the distributed controllers as an example, the remote controller node can maintain a partial view of the network state, based on which the distributed controllers hold distributed dynamic knowledge graphs of the whole network, making it possible to store and query network information in a distributed fashion. In this way, remote controllers work collaboratively, such as figuring out the optimal path selection rule for devices in the data plane by sharing each one's knowledge of the current network state. Via

P4Runtime, the controller then installs the updated path selection rule, i.e., modified table entries, into the P4Runtime Server.

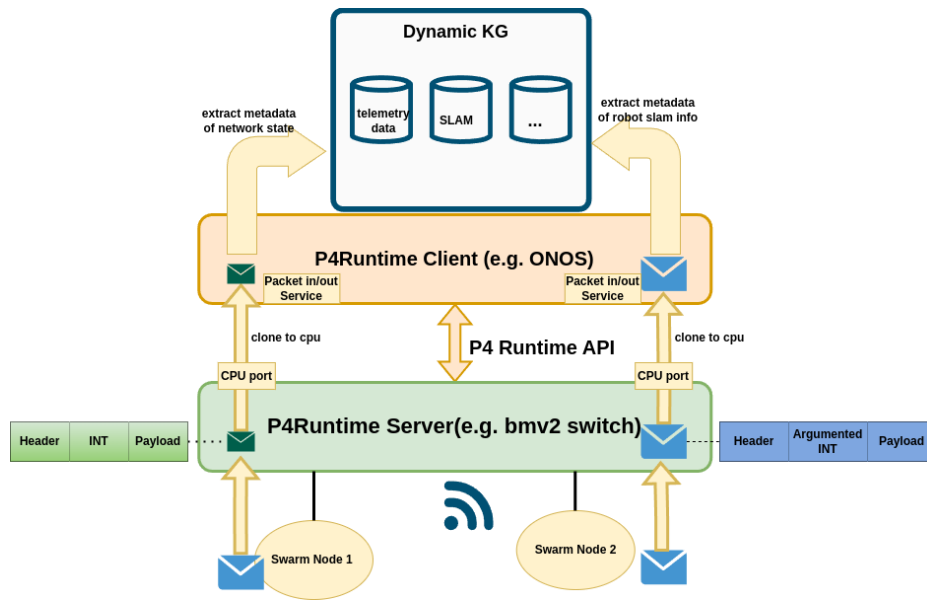


Figure 4-24. RDFizing switch-specific and robot-specific metadata

In Figure 4-25, the shared repository is presented prior to the RDFization process between the Network Control Plane and Middleware Layer, playing a pivotal role in facilitating swarm node management (cf D4.1). The repository comprises two crucial tables: the location and attributes tables. The location table serves to store essential node information, facilitating the management of swarm node joining by maintaining mappings between each node's Access Point ID (AP_ID) and Universally Unique Identifier (UUID). Simultaneously, the attributes table stores additional node details, including mappings between UUIDs and attributes such as Media Access Control Address (MAC_ADDR). These tables collectively provide comprehensive insights into node characteristics and enable efficient coordination and decision-making within the network architecture, laying a solid foundation for seamless integration and management between the Network Control Plane and Middleware Layer.

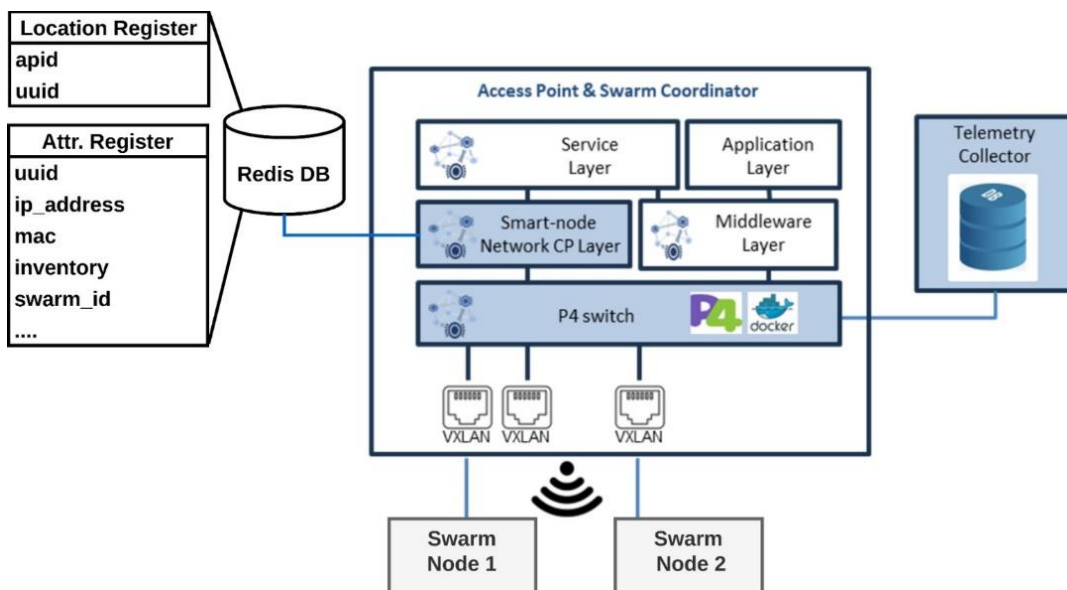


Figure 4-25. Shared repository between Network Control Plane and Middleware Layer

By transforming these entries into RDF and storing in the new DKG, these node-specific information can be integrated into the dynamic knowledge graph. Consequently, various APs located at various P4 controller nodes can exchange node-specific information and answer requests collaboratively.

For instance, in the DDS use case of Figure 4-26, where a publisher publishes a new ROS message only accessible to its subscribers who share the same group ID. As the packet goes through the P4 switch, it will be forwarded to the control plane for further inspection. The controller node would query the swarm nodes that belong to the same group as the publisher and identify those as legal subscribers, and therefore resolve the respective destination addresses from the knowledge graph. With the help of the shared knowledge graph, in this case, the ROS publisher can multicast ROS messages to a specific group of ROS subscribers rather than broadcast messages to irrelevant receivers, realizing the access control for multicasting.

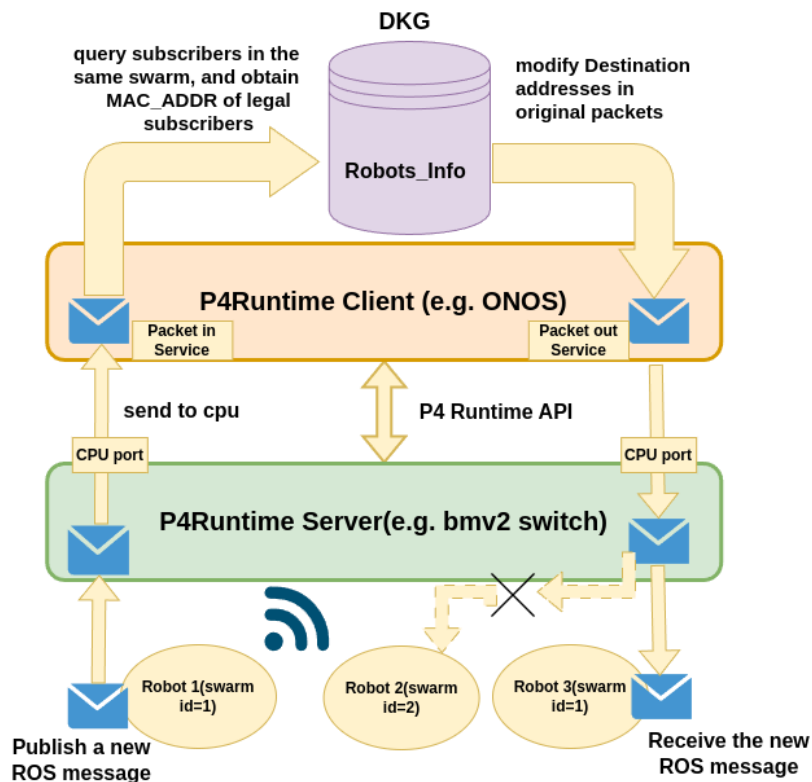


Figure 4-26. Dynamic Knowledge Graph (KG) use case example within DDS messaging framework

4.5.2 Network-aware Swarm Formation with DDS

4.5.2.1 ROS data types and data collections

Robots are designed to collect data through various sensors that allow them to perceive and interact with their environment. These sensors cater to different aspects of perception, such as detecting obstacles, measuring distances, capturing visual information, or sensing physical contact. Examples of sensors include cameras, lidar, radar, ultrasonic, infrared, touch, gyroscopes, accelerometers, and more.

Each sensor type serves a specific purpose, and they continuously measure and capture data from the surrounding environment. For instance, cameras capture images or video feeds, and lidar sensors emit laser beams to create 3D maps of the surroundings. Robot data collection involves a continuous loop of sensing, acquiring, processing, and acting. Integrating advanced

sensors and intelligent algorithms enables robots to navigate, interact with their environment, and perform various tasks autonomously.

The number and type of robot sensors are determined by their design, purpose, and intended tasks. Here are some common sensors used in robotics and how they collect data:

Inertial Measurement Units (IMUs) Sensor: IMUs consist of accelerometers and gyroscopes that measure acceleration and angular velocity, respectively. These sensors help the robot understand its orientation, movement, and changes in velocity.

Camera Sensor: Cameras capture visual information from the environment, and this data is commonly used for tasks such as object recognition, navigation, and mapping. The information gathered by cameras is usually in the form of images or video feeds.

Lidar (Light Detection and Ranging): Lidar sensors work by emitting laser beams and measuring how long it takes for the laser to bounce back after hitting an object. This information is then used to generate highly detailed 3D maps of the robot's surroundings. By analyzing these maps, the robot can navigate and avoid obstacles in its environment more efficiently and accurately.

Force/Torque Sensors: These sensors are essential for force control tasks such as grasping delicate objects or applying specific forces during manipulation. They measure forces and torques applied to the robot's end effector or joints.

Temperature Sensors: It is crucial to monitor the temperature of a robot's components for proper functioning and safety. Sensors can be placed throughout the robot to ensure that critical parts stay within acceptable temperature ranges. Temperature sensors are often installed in edge devices to ensure infrastructure safety.

Microphone Sensor: Robotic devices equipped with microphones can capture audio data, which can be utilized for various purposes, such as speech recognition, environmental sound analysis, and communication.

The control system or onboard computer typically processes the data gathered by a robot's sensors. Advanced algorithms are often utilized to make informed decisions based on this data. This enables the robot to function autonomously or with human guidance, making it a valuable tool in various industries.

4.5.2.2 Swarm coordination mechanisms.

Data Flow: The seamless transfer of data from a physical sensor to a robot and other functional nodes of the robot has been made possible with the aid of the wrapper or sensor APIs and standard message types provided by ROS2. The sensor API effectively converts raw sensor data to a message type that is compatible with ROS2. Subsequently, the ROS2 publisher, using the specific message type, can read sensor data directly from the sensor and publish it through a ROS topic. Furthermore, any functional node can access the published sensor data in a ROS2 message format only by subscribing to the specific topic.

The Intel RealSense camera D435i is equipped with the RealSense SDK and a ROS2-compatible message type wrapper. This wrapper converts raw IMU sensor data to the ROS2 standard message type 'sensor_msgs::Imu', and subsequently publishes it on two different topics - '/camera/camera/gyro/sample' and '/camera/camera/accel/sample'. If the parameter for unite_imu_method is set to a value greater than 0, a new topic called '/{robot_namespace}/imu' is created, containing data from both accel and gyro combined, subsequently published to this

topic. The SmartEdge runtime node can subscribe to the sensor data that the ROS agent has previously published. Once received, the ROS msg to the RDF converter can convert various types of ROS2 messages to RDF data. The converted RDF data is then published through a new topic, which is identified as `"/{robot_namespace}/semantic_imu"`. The semantic stream processing node can subscribe to the `"/semantic_imu"` topic to process the RDF data further. This will allow additional processing to be performed on the data more efficiently and effectively. A similar process is followed for the other sensor data types respectively.

4.5.2.3 Communication at application level – ROS message

ROS utilizes a simplified message description language to describe the data values or messages that ROS nodes publish. This simplification facilitates the automatic generation of source code for various programming languages by ROS tools. The message descriptions are stored in `.msg` files located in the `msg/` subdirectory of a ROS package. The `.msg` file is comprised of two components: fields and constants. Fields represent the data transmitted within the message, while constants define useful values that serve to interpret those fields. Using the built-in data types in Figure 4-27 and message types, it is possible to create custom message types according to the requirement.

Type name	C++	Python	DDS type
bool	bool	builtins.bool	boolean
byte	uint8_t	builtins.bytes*	octet
char	char	builtins.str*	char
float32	float	builtins.float*	float
float64	double	builtins.float*	double
int8	int8_t	builtins.int*	octet
uint8	uint8_t	builtins.int*	octet
int16	int16_t	builtins.int*	short
uint16	uint16_t	builtins.int*	unsigned short
int32	int32_t	builtins.int*	long
uint32	uint32_t	builtins.int*	unsigned long
int64	int64_t	builtins.int*	long long
uint64	uint64_t	builtins.int*	unsigned long long
string	std::string	builtins.str	string
wstring	std::u16string	builtins.str	wstring

Figure 4-27. Built-in types supported by ROS2

4.5.2.4 Semantic ROS for SmartEdge runtime

The design of Semantic ROS (Figure 4-28) consists of three key components, each playing a crucial role in transforming raw ROS data into a semantically enriched RDF format. This not only enhances data interoperability but also opens up new possibilities for advanced data processing and querying.

ROS Scanner Node: The first component is the ROS Scanner Node; it acts as a discovery agent within the ROS network. Its primary function is to scan the network, identify all available data topics, and categorize them based on their types. This information serves as the foundation for the subsequent components, ensuring a comprehensive understanding of the existing data landscape.

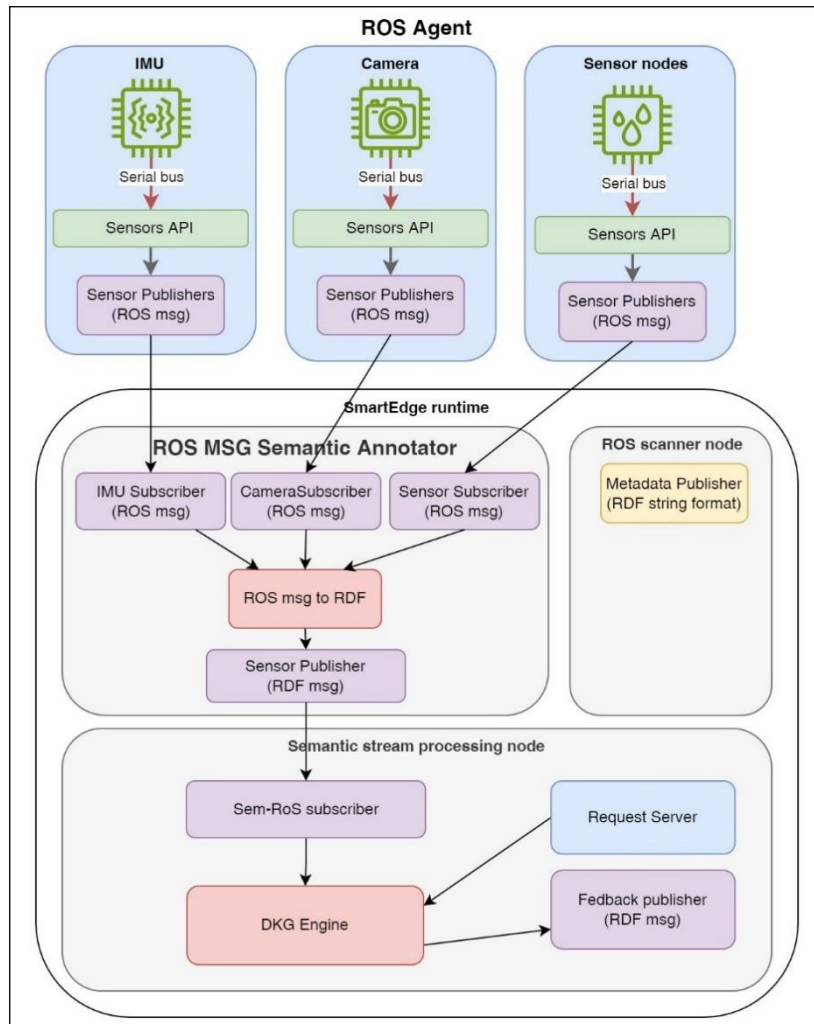


Figure 4-28. Semantic ROS design

ROS Message Semantic Annotator: Building on the data catalog provided by the ROS Scanner Node, the ROS Message Semantic Annotator takes on the task of transforming raw ROS messages into RDF format. This component subscribes to data topics generated by various ROS nodes, such as cameras, IMUs, and laser scanners. It then applies semantic annotations based on a predefined ontology, converting the data into RDF and publishing it as new ROS topics. This transformation not only adds a layer of semantic richness to the data but also facilitates easier integration with other semantic web technologies.

Semantic Stream Processing Node: The final component, the Semantic Stream Processing Node, operates as a ROS action server and interfaces with a Dynamic Knowledge Graph engine. This component subscribes to the RDFized data produced by the ROS Message Semantic Annotator and provides a platform for querying this data using SPARQL queries and receive the results via feedback publisher over DDS. This allows robots within the SmartEdge swarm to request specific

information based on their needs, opening possibilities for dynamic and intelligent decision-making.

4.6 ORCHESTRATION VIA CONTINUOUS QUERY FEDERATION

T5.3 introduces the Continuous Query Federation (CQF) mechanism for the Orchestrator to continuously federate a query to a network of interconnected nodes. This allows for dynamic and collaborative data processing among SmartEdge devices within SmartEdge Swarms, functioning as autonomous processing agents (a SmartEdge node or a Coordinator representing a swarm). In essence, an Orchestrator can delegate partial workloads to their peers through subscribing to continuous queries using following operations:

- **Subscription and Discovery Operations:** The mechanism involves subscription and discovery operations to enable the deployment of a stream processing pipeline across distributed sites. This is particularly useful for scenarios where data streams are generated from different locations, such as cameras, lidar streams from vehicles or robots.
- **Autonomous SmartEdge Nodes:** Each autonomous SmartEdge node serves as a processing node and provides access to data streams. These streams can originate from various sensors, and the node can partially process the incoming data before forwarding the results as mappings or RDF stream elements to its parent node.
- **Query Federation:** When a query is subscribed to the top-most node (root node), the query is divided into sub-query fragments. These fragments are then deployed at one or more sites via subscribed nodes. The query fragments, consisting of one or more operators, can be distributed across different processing nodes. The federation process is recursive, allowing sub-queries to be delegated to subscribed nodes.
- **Synchronization and Timing Stream:** All participant nodes in the processing pipeline synchronize their processing timeline using a timing stream propagated from the root node. This ensures that the processing of sub-query fragments is coordinated across the network.
- **Dynamic Processing Topology:** The federation process is dynamic and can be adjusted in real-time. The processing topology of the pipelines can be configured dynamically by changing how and where participant nodes subscribe to the processing networks.

To enable implement such a federation mechanism, we extend a decentralized version of the CQELS engine [Danh11] named Fed4Edge [Manh19]. Thanks to the platform-agnostic design of its execution framework [Danh15], the core components are abstract enough to seamlessly integrate with various RDF libraries, facilitating portability across different hardware platforms.

The core components for CQF is depicted in Figure 4-29 CQL reuse the core components of CQELS and RDF4Led, such as the Dictionary, Encoder, Decoder, Dynamic Execution, Adaptive Query Optimizer, and Buffer Manager. Additionally, extension plugins, such as Adaptive Federator, Thing Directory, Stream Input Handler, and Stream Output Handler, are built to facilitate the federation mechanism.

The CQF architecture seamlessly integrates components from CQELS and RDF4Led, optimizing RDF data processing for edge devices. At its core, the Encoder, Decoder, and Dictionary collaborate to normalize RDF data, reducing memory usage and enhancing cache efficiency [Anh18]. This normalized data flows through the Stream Input Handler, where it is encoded before being passed to the Buffer Manager for further processing.

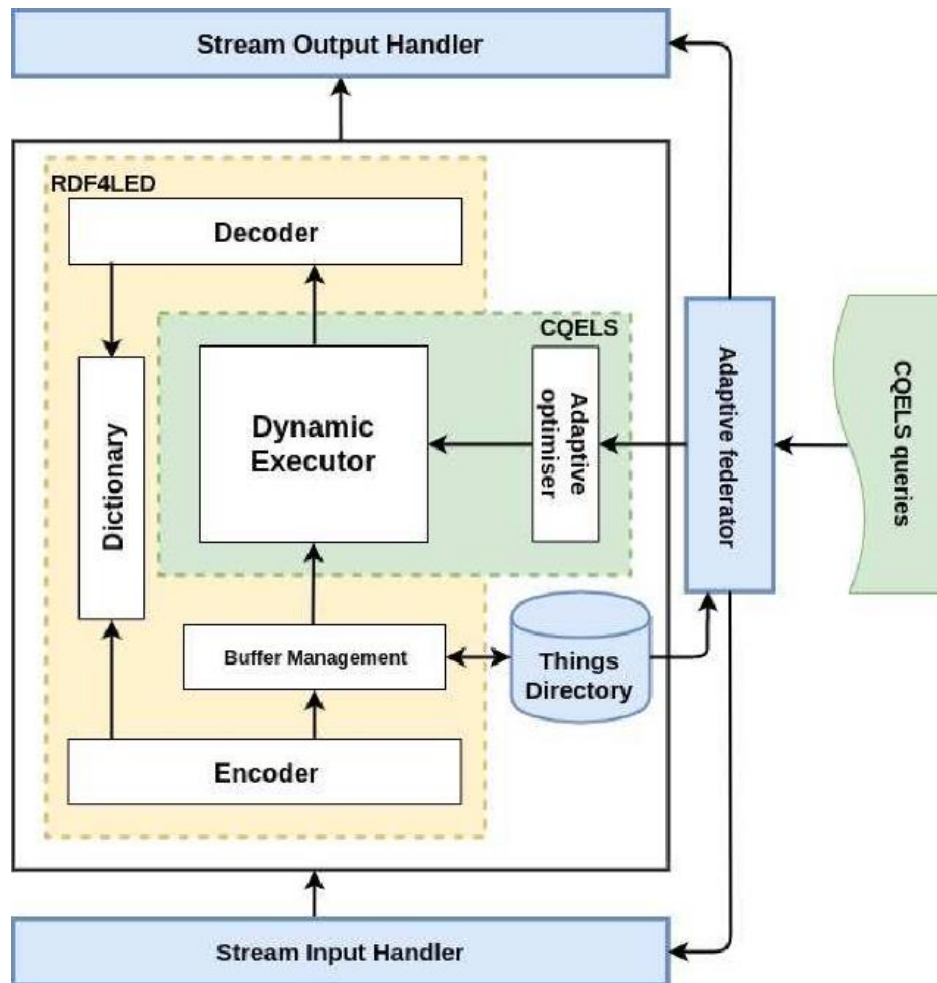


Figure 4-29. Key components to enable Continuous Query Federation

The Buffer Manager plays a pivotal role in managing windowed data and cached information, utilizing flash-aware updating algorithms for swift data updates [Danh11] [Danh17]. From here, the data proceeds to the Dynamic Executor, which dynamically adjusts query execution strategies based on the evolving data distribution [Danh18].

Driving optimization further, the Adaptive Optimiser continually refines query plans, leveraging the insights gained from static data patterns. This optimization process ensures efficient processing, whether employing fresh or incremental update policies for join results handled within the buffer. When a task is offloaded to a special node with hardware acceleration, the optimizer calls the runtime optimizer of T5.2 (see above) to optimize the hardware acceleration part of the operation.

The Adaptive Federator serves as the engine's query rewriter, intelligently dividing queries into subqueries and pushing operators close to streaming nodes for enhanced performance. Complementing this, the Thing Directory stores metadata for service discovery, fostering seamless communication between SmartEdge nodes[Dell17].

Throughout this process, the Stream Input Handler subscribes to and listens for subquery results, while the Stream Output Handler efficiently disseminates these results to other nodes or back to the requester.

Overall, the interconnectedness of these components forms a cohesive architecture tailored for efficient and adaptable RDF data processing in SmartEdge environments.

4.6.1 Preliminary Experiment and Results

From our work in [Manh19], we implemented the baseline CQF to profile and analyse performance behaviour of the systems at the IoT-Edge-Cloud Testbed at TU Berlin.

Evaluation Setup

Hardware & Software: The experiment hardware consists of a cluster comprising 85 Raspberry Pi model B nodes. Each node boasts a Quad-Core 1.2GHz Broadcom BCM2837 64bit CPU, 1GB RAM, and 100 Mbps Ethernet capability. These nodes are interconnected through five TP-LINK Jetstream T2500-28TC switches, each featuring 24 100 Mbps Ethernet ports and 4 1000 Mbps uplinks. The T2500-28TC switch offers a non-blocking aggregated bandwidth of 12.8Gbps. Four of these switches, responsible for connecting streaming nodes, are linked to the fifth switch via the 1000 Mbps links. The fifth switch facilitates connections to the CQELS processing nodes. All nodes operate on Raspbian Jessie as the operating system, utilizing OpenJDK 1.7 for ARM as the JVM. We've configured a maximum heap size of 512MB for the Fed4Edge engine on each node.

Experiments and Results

Baseline Calibration (Exp1): In the first experiment, we calibrated the maximum processing capability of a node, using Raspberry Pi devices as edge nodes and a high-performance PC as a server. We identified a bottleneck phenomenon, revealing that increasing the number of streaming nodes beyond four decreased overall processing throughputs due to bandwidth limitations.

In this experiment, we calibrated the maximum processing capability of a processing node as the baseline for the following federation experiment. We increased the number of stream nodes to observe the bottleneck phenomena whereby increasing more streaming nodes decreases the processing throughput of the network. Each streaming node will stream out recorded data as its maximum capacity. We will use Query 1 and its two variants as the testing queries. These two variants are made by reducing four triple patterns into 1 and 2 patterns respectively. The throughput is measured by using a timing stream whereby each streaming nodes will send timing triples indicating when each of them starts and finishes sending their data. In each test we will equally split 500k-1M observations among streaming nodes and record how much time to process these observations to calculate the average throughput. Note that we separated the streaming and processing processes in different physical devices to avoid performance and bandwidth interference which might have an impact on our measurements.

Figure 4-30 the results of the experiment Exp1. The maximum processing throughput for three variants of Query 1 on one single edge device is from 4200-5000 triples corresponding to 4 streaming nodes. It is interesting that increasing the number of streaming nodes more than 4 will gradually decrease the overall processing throughput. The results are consistent with different complexities of the variants of Query 1. We observed that the CPU usages were around 60-70% and the memory consumption was around 270-300MB in all tests. Therefore, we can conclude that the bottleneck was caused by bandwidth limitations. We also carried out a similar test with Q1 on a PC (Intel Core i7 i7-7700K, 4GHz, 1GBb Ethernet and 16GB RAM) as the root node which has more than 10 times of processing power, memory and network bandwidth than those of a Raspberry Pi model B. As we expected, the PC's maximum throughput is

approximately 36k triples/second, around 8-10 times the one with a Raspberry Pi. Note that this PC costs more than the price of 40 Raspberry Pi nodes.

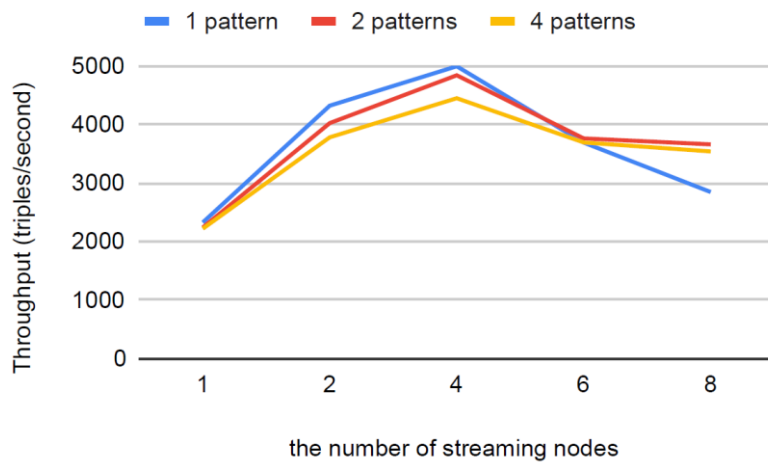


Figure 4-30. Baseline experiment result

Fan-out Federation (Exp2)

The second experiment explored the concept of fan-out federation, assessing the impact of increasing the number of edge nodes on processing throughput.

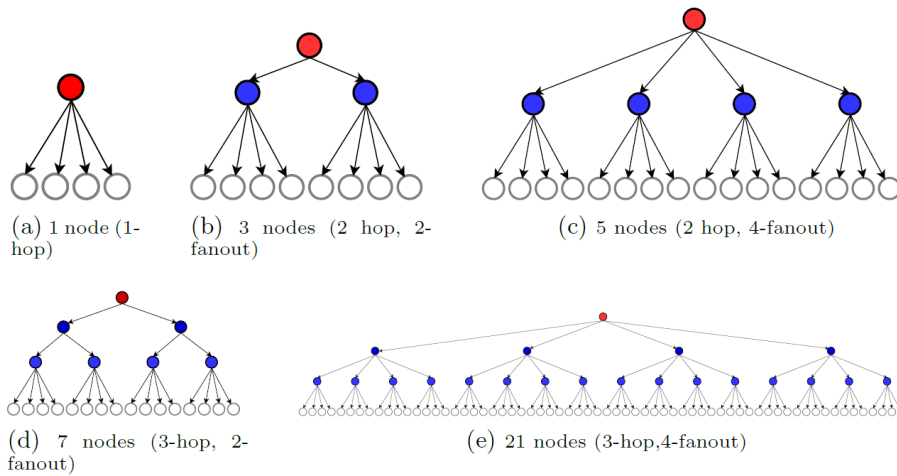


Figure 4-31. Topology

To test the possibility of increasing the processing throughput by increasing more edge nodes as autonomous agents to the network, we carried out the tests on five topologies as shown in Figure 4-31. The first topology (1-hop) in Figure 4-31 (a) was the configuration that gave the peak throughput in Exp1. Let k be the number of hops the data has travelled to reach the final destination, we will increase k to add more intermediate nodes to this topology to create new topologies. As a result, we can recursively add n nodes to the root node (k=2, namely 2-hop) and then n nodes to the root node's children nodes (k=3, namely 3-hop) whereby n is called the fanout factor (denoted as n-fanout). Then, we have $\sum_{i=0}^{k-1} n^i$ as the number of nodes of a topology with k hops and fanout factor n. We choose n=2 and n=4 (corresponding to the number of streaming nodes at the maximum throughput reported in Exp1 below), thus, we have four new topologies with 3, 5, 7 and 21 processing nodes in Figure 4-31 (b), Figure 4-31 (c), Figure

4-31 (d) and Figure 4-31(e). In each processing topology, the lowest processing nodes are connected with 4 streaming nodes. We will record the throughput and delay for processing three queries (Q1, Q2, Q3 and Q4) on these five topologies in a similar fashion to Exp1.

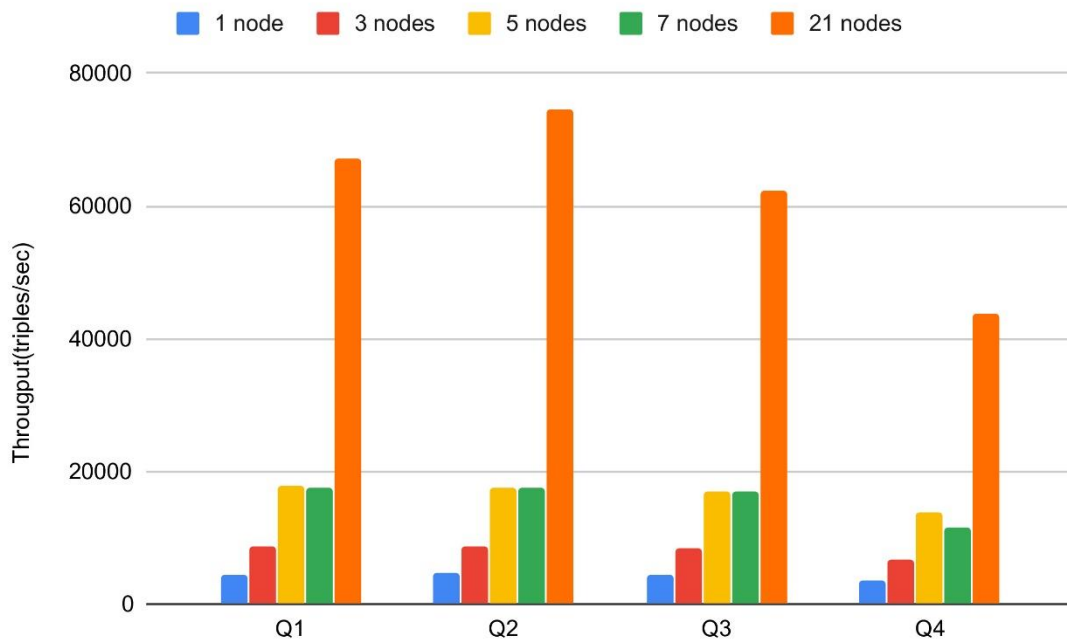


Figure 4-32. Fan-out Federation Experiments: Processing Throughput Results

Figure 4-32 shows the results of throughput improvements via federating the processing workload on other intermediate nodes in four proposed topologies. The results show that adding more nodes will increase the processing throughput in general. Most queries have their processing throughput consistently boosted up as a considerable amount of processing load was done at the intermediate nodes. However, the increase is not consistently correlated with the total number of processing nodes. In fact, the topology with 5 nodes in Figure 4d gives a slightly higher throughput than those of the topology with 7 nodes in Figure 4c. This can be explained by the fact that both topologies have 4 processing nodes at the lowest levels (called leaf processing nodes, i.e., connecting to streaming nodes) but the data in the latter topology has to travel 1 more hop in comparison with the former. Due to our pushing down, rewriting strategy presented in Section 3, these two upper blue nodes in Figure 4c did not significantly contribute to the overall throughput but on the other hand cause more communication overhead.

Looking closer at the reported figures, we see a high correlation between the number of leaf processing nodes, i.e. n^{k-1} , and the processing throughput in all topologies. This shows that our proposed approach is able to linearly scale a network of IoT devices by adding more devices on demand. In particular, a network of 21 Raspberry Pi nodes can collaboratively process up to 74k triples/seconds or equivalent to roughly 8500 sensor observations/second that are streamed from other 64 streaming nodes. Hence, the above 20K weather stations across the globe of NCDC dataset can be queried via such a network with the update rate 20-30 observations per minute which are much faster than the highest up- date rate currently supported by NCDC 6, i.e. ASOS 1-minute data. Moreover, the processing capacity of this network is twice as much as that of the above PC but it only costs roughly half of the PC. Regarding the energy consumption, each Raspberry Pi only consumes around 2W in comparison of 240W of the above PC.

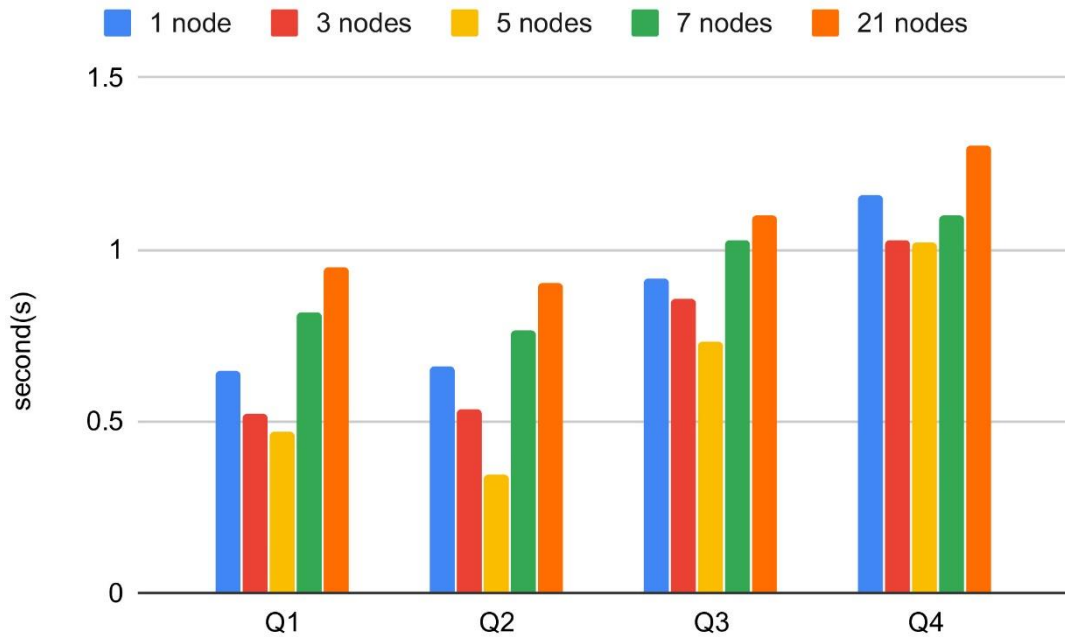


Figure 4-33. Fan-out Federation Experiments: Average Processing Time Results

Figure 4-33 reports the average time for each observation to travel through a processing pipeline specified by each query on different topologies, i.e., average processing time. It shows that adding more intermediate nodes for query Q1 and Q2 can lower the average processing time as it can reduce queuing time at some nodes. That means communication time might be a dominant factor for the delay in these processing pipelines. In queries Q3 and Q4, we witness the consistent increase in processing time w.r.t. the number of hops which explains the nature of query Q3 and Q4 that needs more coordination among nodes. However, it is interesting that increasing 1 hop in organizing a network topology just adds 10-15% delay while the maximum throughput gain is linear to n^{k-1} .

4.6.2 From empirical insights to design and implementation of Orchestrator and Optimizer

We observed the CPU, memory consumption and bandwidth in our experiments. It is interesting that all tests used 60-70% of CPU (across 4 cores), 25-30% of physical memory and 20-40% of Ethernet bandwidth (i.e., 100Mbps). Our reported performance figures show that edge devices have enough resources to enable semantic interoperability for the edge computing paradigm. From our analyses of hardware and software libraries, the most potential suspects for the processing bottleneck are related to the communication among the nodes. Hence, there is a lot of room to make the Orchestrator much more efficient and scalable. In this context, to achieve KPI K4.3 and K4.5, we are going to address following challenges.

The first challenge is how to address the multiple optimization problems that such a federated processing pipeline entails. The first one is how to optimize an RSP engine for edge devices which have distinctive processing and I/O behaviors from those of PC/workstations due to their own design philosophies. The second challenge is about how to find optimal operator placements on very dynamic execution settings. The subsequent challenge is how to define cost models which

are no longer limited to processing time/throughput but need to cover several cost metrics such as bandwidth, power consumption and robustness.

Regarding cooperation and negotiation among RSP autonomous agents, the next challenge is the study and exploration of protocols and strategies that follow the multi-agent system paradigm. Although early works on the topic [Tommasini19] point at potential opportunities in this area, several aspects have not been studied yet. These include the usage of individual contextual knowledge for local decision making (potentially through reasoning) and for a resource-optimised distribution of tasks among a set of competing/associated nodes. The dynamics of these federated processing networks would need to adapt to changing conditions of load, membership, throughput, and other criteria, with emerging behaviour patterns on the sensing and processing nodes.

5 CROSS LAYER TOOLCHAIN FOR DEVICE-EDGE-CLOUD CONTINUUM

5.1 OVERVIEW OF CROSS-LAYER TOOL CHAIN FOR DEVICE-EDGE-CLOUD CONTINUUM

T5.4 will integrate components from network layer to application layer into an integrated toolchain so that it can build a runtime toolchain to abstract the capabilities of sensors, Edges and Cloud using a declarative programming model. This model will integrate the network programmability of WP4 and semantic interoperability of WP3 with semantic specifications of such capabilities. Via these semantic specifications, the tool chain will release a developer from having to specify device configurations, sensory inputs, network settings and data operations of data stream processing pipelines or application logic in various programming languages and configuration formats.

First T5.4 will develop the runtime toolchain that can read a high-level specification to prepare the necessary runtimes for SmartEdge nodes to be deployed for certain application settings. The task will extend the compiler SSR[LePhuoc2021] which was developed by TUB to read specifications written in grammar composed from SPARQL or SHACL. For instance, to prepare a deployment including a set of RSUs in UC2, the developer will prepare a configuration file in RDF (e.g, JSON-LD or RDF-Turtle) for such deployment, which includes description of hardware configurations of edge computing units and connected sensors, e.g., camera, LiDARs and other metadata related the deployment context. Note that a visual editor can be developed to provide a wizard-based GUI to guide the developer on the step-by-step process. Also, reusable templates or previous configurations can be utilized with the support of SPARQL features from the underlying knowledge graph engine.

From the semantic specification of a deployment, the runtime toolchain will build, and pack necessary components provided in WP3, WP4, T5.1, T5.2 and T5.3 into runtime artifacts to install or deploy into SmartEdge nodes involved in the deployment. Some examples of such artifacts would be docker images, ROS packages, Linux/Window sandboxes or Jar packages for JVMs. Note that such an artifact will contain a bootstrapping knowledge graph to describe the smart-node, such as expected roles and capabilities, so that it has enough data to join or to form a swarm autonomously.

Another advanced feature for such semantic specification is pre-configure a deployment to serve a task or recipe in WP3 by including them in its deployment context metadata. For example, the developer would only have to specify semantic outputs of Semantic SLAM maps in Smart Factory UC or semantic queries over high-level objects and events in automotive UCs in standardized formats/languages to connect them via low-code editor of WP3 to generate a recipe to specify the application logic. Then, the runtime toolchain will have to compute how to inject the coordination logic into the Federator of T5.3 to form a swarm of participant nodes, to setup data links, and how to route the data to AI components according to their sensing and processing capabilities, in order to be able to generate such high-level expected outcomes.

By integrating features provided by Federator and Optimizer of T5.3, the runtime toolchain will direct the participant SmartEdge nodes to dynamically load and configure all necessary components into a decentralized application logic to participating IoT devices, edge and cloud. It will rely on the features of Federator T5.3 to decompose the query to necessary data and AI operations to generate federated execution plans to generate the desired outputs. Besides, such an execution plan will be adaptively coordinated and optimized with the mechanism in T5.3 with the capability to elastically scale on demand in T5.2.

5.2 REQUIREMENTS

ID/Ver: SW-018/v1.1	Related Use Case(s): UC-3	Task: T5.4
<p>A swarm must be able to handle smart nodes by mapping their data into the same coordinate system regardless of their movement. For example, for two smart nodes in a swarm to cooperate about the physical movement of an object, they must have the same special frame of reference, or be able to transform the other smart-nodes frame of reference into their own. The same holds true for temporal mappings, i.e., all collaborating smart nodes must share a common concept of time.</p>		
<p>The Semantic Programming model provided by T5.4 and the Data Fusion Component provided by T5.1 will enable these computations.</p>		

ID/Ver: SW-025/v1.1	Related Use Case(s): UC-3, UC-4	Task: T5.4
<p>The swarm can be made up of homogeneous or heterogeneous swarm nodes and capabilities. The swarm node can all be of the same type and capabilities, or they can be of different types and capabilities.</p>		
<p>The Semantic Programming model provided by T5.4 will allow interoperability and address these heterogeneities.</p>		

ID/Ver: HP-003/v1.1	Related Use Case(s): UC-2, UC-3, UC-5	Task: T5.4
<p>Capability to read radar message format (JSON stream).</p>		
ID/Ver: HP-004/v1.1	Related Use Case(s): UC-2,	Task: T5.4
<p>Capability of reading and processing camera stream data (currently RTSP).</p>		
ID/Ver: HP-005/v1.1	Related Use Case(s): UC-2,	Task: T5.4
<p>Capability to read Controller status messages (JSON stream).</p>		
ID/Ver: HP-008/v1.1	Related Use Case(s): UC-2,	Task: T5.4
<p>Ability to handle public transit open data (tram locations) from outside.</p>		
ID/Ver: HP-009/v1.1	Related Use Case(s): UC-2	Task: T5.4
<p>Support for Helsinki's open data API for providing data to Helsinki from the swarm sensors.</p>		
ID/Ver: HP-010/v1.1	Related Use Case(s): UC-2, UC-3, UC-5	Task: T5.4
<p>Dashboard or user interface (web client) for viewing the data (sensor network, traffic indicators, swarm operation, raw data).</p>		
<p>Several SmartEdge plugins provided by T5.4 will fulfill these requirements.</p>		

ID/Ver: HP-013/v1.1	Related Use Case(s): UC-2, UC-3, UC-5	Task: T5.4
<p>The SmartEdge components system must be able to run on ARM-based hardware and Ubuntu/Linux OS.</p>		
ID/Ver: HP-015/v1.2	Related Use Case(s): UC-2, UC-3, UC-5	Task: T5.4
<p>The SmartEdge components system must be able to run on x86-based hardware and Ubuntu/Linux OS.</p>		
<p>SmartEdge Runtime will be implemented to be portable to different hardware or OS.</p>		

ID/Ver: HP-016/v1.2	Related Use Case(s): UC-2, UC-3	Task: T5.4
It should be possible to dynamically add a new sensor stream to an existing smart-node, as sensors on other devices may join or leave the swarm.		
T5.4 SmartEdge Runtime will detect and update to the DKG if new sensors are found.		

ID/Ver: LC-001/v1.2	Related Use Case(s): UC-2, UC-3, UC-4	Task: T5.4
By default, SmartEdge nodes will use standardized data model of RDF for representing intermediate results and data exchange		
The toolchain is designed to process semantic data using RDF data model.		

ID/Ver: LC-002/v1.2	Related Use Case(s): UC-2, UC-3, UC-4, UC-5	Task: T5.4
The low-code programming toolchain should provide a DSL to programmatically generate some basic template for building Recipes towards drag and drop functionality and minimal programming.		
ID/Ver: LC-003/v1.2	Related Use Case(s): UC-2, UC-3, UC-4	Task: T5.4
The low-code programming toolchain should be able to store the created Recipes in RDF format		
ID/Ver: LC-010/v1.1	Related Use Case(s): UC-2, UC-3, UC4	Task: T5.4
Low-code platform should support code generation required for execution of a recipe (both interactions between the devices/communication, and device's logic).		
Mendix plugins provided by T5.4 will fulfill these requirements.		

ID/Ver: LC-011/v1.1	Related Use Case(s): UC-2, UC-3, UC-5	Task: 5.4
Functionality should be configurable for filtering data/creating events by using stream from single node. For example, option zone safety functionality (number of vehicles in certain area at certain time) should be easily configured to new nodes, or health biomarker data stream by the logical indicator processor.		
ID/Ver: LC-012/v1.1	Related Use Case(s): UC-2, UC-3	Task: T5.4
Functionality should be configurable for filtering data/creating events by using stream from multiple devices. For example, red runner detections should be configurable to devices (combines data from signal controller and radar). This would be an extension of LC-011 (only operating in single node)		
ID/Ver: LC-013/v1.1	Related Use Case(s): UC-2	Task: T5.4
There should be the possibility for configuring automatically/semi automatically parameters for device sensors and how to perform the functionality such as LC-011 and LC-012 and other simple manipulation of data.		
T5.4 will implement stream query operator as SmartEdge primitive which will do the computation required by these requirements.		

ID/Ver: LC-016/v1.1	Related Use Case(s): UC-3, UC-5	Task: T5.3 & T.54
----------------------------	--	--------------------------

The low-code platform should provide a mechanism to define recipes that will be instantiated as applications and run by one of more smart-nodes in the swarm.

T5.4 provides a Semantic Programming model that allows to describe recipes as semantic programs. The orchestrator will decide if the program will be run on one smart node or several smart nodes.

ID/Ver: LC-024v2.1	Related Use Case(s): UC-1, UC-2, UC-3, UC-4, UC-5	Task: 5.4
SmartEdge should provide a user interface that allows a user to check the current state. For example, available nodes and available sensors and actuators including their available status and value ranges.		
Users can check the current state by sending query the DKG of any smart node of a swarm.		

ID/Ver: CSI-001/v1.1	Related Use Case(s): UC-1, UC-2, UC-3, UC-4, UC-5	Task: T5.4
SmartEdge must provide a mechanism so that (swarm) devices (e.g. vehicles) can receive information from the environment.		
SmartEdge Runtime will collect information from environment		

ID/Ver: CSI-006/v1.1	Related Use Case(s): UC-1, UC-3	Task: T5.4
SmartEdge must provide a semantic reasoner for verification of situations in video stills or streams with respect to applicable rules that must be followed based on the detected situations.		
The semantic program resolver will work a reasoner.		

ID/Ver: CSI-007/v1.1	Related Use Case(s): UC-2, UC-3, UC-5	Task: T5.4
Information about all the available data sources/sensors and actuators present should be available at all times. This is somewhat similar to SW-007 for edge nodes, but in this case, we are talking about units within each node.		

ID/Ver: CSI-008/v1.1	Related Use Case(s): UC-2, UC-3, UC-5	Task: T5.4
We should have availability of static information about the environment in standardized format. That is, there should be a way to check physical parameters in the field (e.g., in UC-2 the location of lanes, what is their logical connection, what lanes are controlled by what signal heads, in UC-5 the status and location of a person indoors/outdoors, air quality).		

ID/Ver: CSI-009/v1.1	Related Use Case(s): UC-2	Task: T5.4
As extension of CSI-008 regarding variable environmental data (e.g., how long are the queues, the detected vehicles represent) for detected objects and other dynamic data (traffic controller statuses) have to be available at all times.		
The information will be described in RDF and maintained in DKG as presented in Section 5.4.1		

ID/Ver: CSI-020/v1.2	Related Use Case(s): UC-3	Task: T5.4
By virtue of CSI-018 a recipe must know the primitives necessary to execute the application on a swarm. A mechanism must exist to match up the primitives to the characteristics of possible		

nodes in the swarm, and in this way define the types of nodes that will be required by a swarm to execute an application.

The semantic program will include information of the necessary primitives to execute the program.

ID/Ver: LC-022/v1.1	Related Use Case(s): UC-2	Task: 5.4
There should be an interface for devices for configuring/adjusting parameters at run time		
The SmartEdge Runtime will allow the reconfiguration parameters for SmartEdge primitives.		

ID/Ver: CSI-026/v2.1	Related Use Case(s): UC-1	Task: 5.4
It must be possible to semantically bind physical devices with virtual devices. (For example, to represent a digital twin with its real-world counterpart including its state).		
Several SmartEdge connectors and adapters will allow this binding feature		

ID/Ver: CSI-027/v1.1	Related Use Case(s): UC-1	Task: 5.4
A binding between real devices and 3D Assets which are used in virtual environments should be possible to automatically react to state changes of the asset or the real device (low code).		
Several SmartEdge connectors and adapters will allow this interaction.		

5.3 PRELIMINARIES AND STAGE OF THE ART

5.3.1 Object Detections for Edge Devices

In the context of D5.1, object detection is the core model to localize the objects in the video scenes to build scene understanding in T5.1. With the focus on making object detection, and data fusion in general, to work on edge device, this section delves into the realm of object detection on edge devices, with a specific focus on profiling the inference speed of Deep Neural Networks (DDN) deployed on the NVIDIA Jetson family. The profiling of these devices, encompassing the Jetson Nano, TX2, Xavier, and Jetson Orin, becomes a crucial basis of achieving the KPI K4.1 (also related to K4.2 and K4.5) and are integral to the successful integration of NVIDIA's Jetson family, serving as key components in the execution of Use Cases 2 and 3 (UC2 and UC3) within the project. Meanwhile, Object detection is an important computer vision task with applications in autonomous vehicles, surveillance, robotics, etc. Performing accurate object detection on resource-constrained edge devices, such as jetson series devices, is challenging but critical for deploying these applications. Based on this, D5.1 aims to contribute actionable knowledge for the strategic utilization of these devices in the project's real-world applications.



	Jetson Nano	Jetson TX2	Jetson Xavier	Jetson Orin
				
CPU	4-Core ARM A57 @ 1.43GHz	4-Core ARM A57 @2.0 GHz 2-Core Denver2 @2.0GHz	8-Core Carmel ARM v8.2 @2.26 GHz	12-Core ARM-A78AE @2.2 GHz
RAM	4GB 64-bit LPDDR4 1600 MHz 25.6 GB/s	8GB 128-bit LPDDR4 1866 MHz 59.7 GB/s	32 GB 256-bit LPDDR4x 2133 MHz 136 GB/s	64GB 256-bit LPDDR5 3200 MHz 204.8 GB/s
GPU	128-core Maxwell GPU 4GB vRAM 472 GFLOPS	256-core Pascal GPU x8GB vRAM 1.33 TFLOPS	512-core Volta GPU 32GB vRAM 32 TOPS	2048-core Ampere GPU 64 GB vRAM 275 TOPS

Table 1. List of Technical Specifications for Jetson series devices.

The Jetson series contains embedded systems with varying levels of computing capability, power consumption, and form factors. As shown in Table 1, the Jetson Nano is the lowest power device at 5-10 Watts, followed by the Jetson TX2 at 7.5-15 Watts. The Jetson AGX Xavier has much greater performance with 30-32 TOPS but higher power consumption from 10-30 Watts. Hence, there is a tradeoff between power and performance based on the specific application requirements.

In object detection, 2D object detectors take images as input and output 2D bounding boxes with classwise predictions. They can serve as real-time detectors but are limited to the image plane and usually without environmental depth perception. In contrast, 3D object detectors take point clouds from sensors like LiDAR as input and output 3D bounding boxes enabling a richer perception of the environment. Most frameworks utilize DNN consisting of feature extraction layers followed by prediction heads. Two-stage detectors like Faster R-CNN (FRCNN) (2D) and PointRCNN (3D) have higher accuracy but are slower compared to one-stage detectors like YOLO (2D) and PointPillars (3D) which prioritize speed. Besides, one-stage detectors (YOLO, DETR) now have precision rivaling two-stage models (FRCNN, SSD). However, both approaches incur significant computational expenses unsuitable for resource-constrained edge devices without optimization. Hence, the direction to decrease the power consumption including neural architecture search (NAS) to find optimized networks, model compression via quantization and pruning, hardware-aware optimization using specialized AI accelerators, and leveraging 2D detectors to generate 3D detections without expensive 3D convolutions, enabling real-time performance.

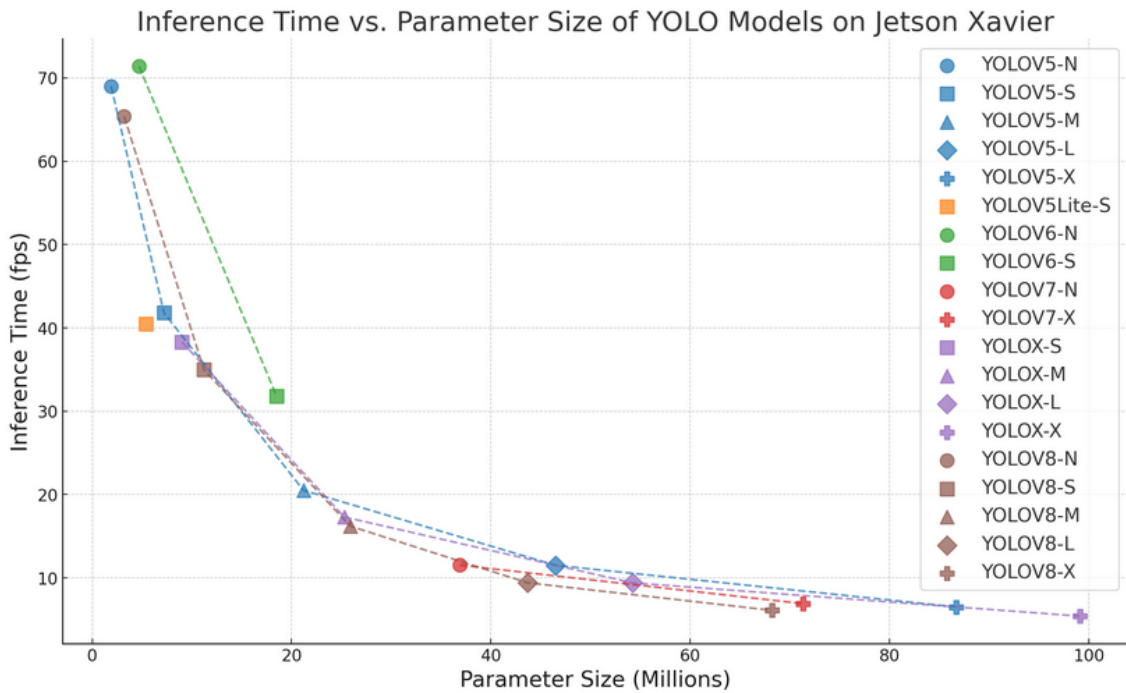


Figure 5-1 Profiling YOLO-series models on Jetson AGX Xavier.

To investigate the performances of jetson series devices on detection models, we profiled YOLO series models on Jetson AGX Xavier, the experimental setup involved a consistent input resolution of 640x640 pixels. For each model, a warm-up phase of 50 iterations was conducted, followed by 1000 iterations to assess performance. The metrics of interest were frames per second (fps) to gauge inference speed and the number of parameters (in millions, M) to investigate how the model complexity will influence the inference speed on edge devices.

As shown in Figure 5-1, in terms of inference speed, YOLOV6 led with 71.4 fps, closely followed by YOLOV8 at 65.4 fps. These models showed an excellent balance of inference speed and efficiency, particularly in their Normal size (N). Besides, YOLOV5, a widely recognized variant of YOLO series model, performed reasonably well, with its N variant achieving 69.0 fps. In contrast, YOLOV7 exhibited the lowest fps rates, with YOLOV7-N achieving only 11.52 fps, and its YOLOV7-X variant at 6.9 fps, indicating a significant trade-off between complexity and speed.

Params (mAP) Model	N	S	M	L	X
YOLOV5	1.9M (28.0)	7.2M (37.4)	21.2M (45.4)	46.5M (49.0)	86.7M (50.7)
YOLOV5Lite	-	5.4M (39.1)	-	-	-
YOLOV6	-	4.7M (37.5)	18.5M (45.0)	-	-
YOLOV7	-	-	-	36.9M (51.4)	71.3M (53.1)
YOLOX	-	9.0M (40.5)	25.3M (46.5)	54.2M (49.7M)	99.1M (51.1)
YOLOV8	3.2M (37.3)	11.2M (44.9)	25.9M (50.2)	43.7M (52.9)	68.2M (53.9)

Table 2 Model Size and Performance on MS-COCO [Lin14] Benchmark.

Regarding parameter size, YOLOV8-N was the most parameter-efficient with only 3.2M parameters, while YOLOX-X was with 99.1M parameters. It is noteworthy that YOLOV8 not only excelled in inference speed but also maintained a lower parameter count across all variants, suggesting an optimized architecture for edge devices like the Jetson Xavier. It indicates that there is a general trade-off between inference speed and model complexity across different YOLO versions and their variants. Light models with fewer parameters tend to infer faster, as seen in YOLOV6-N and YOLOV8-N, making them more suitable for real-time applications where latency is crucial. On the other hand, models with more parameters, like YOLOV8-L and YOLOV8-L-X, are better suited for scenarios where accuracy is prioritized over inference speed. More details about model sizes and their corresponding performances on MS-COCO [Lin14] benchmark is shown in Table 2.

Hence, the choice of a deployable model on Jetson series edge devices such as Jetson Xavier should be primarily guided by the specific requirements for inference speed versus model complexity, devices computing power. Models like YOLOV6 and YOLOV8 appear to offer a more balanced profile for edge computing environments, providing a good compromise between inference speed and model size.

5.3.2 Preliminary results for Semantic Programming for edge computing

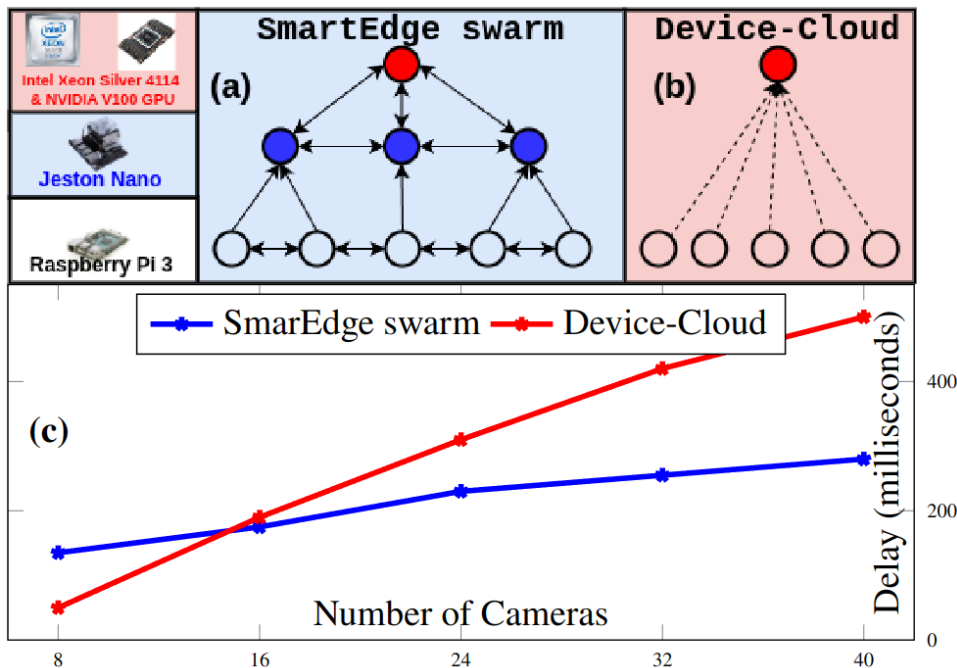


Figure 5-2 Profiling edge performance according to diver number of cameras.

This section presents the initial implementation of our case study on a distributed camera network aimed towards UC2. This network is constructed based on the data provided by the AI City Challenge (AIC) [Naphade19]. The AIC encompasses 40 cameras spanning across 10 intersections in a mid-sized US city. The baseline used for comparing the efficiency and scalability of the SmartEdge swarm is the traditional device-cloud infrastructure. This infrastructure ingests data from the 40 cameras connected by 40 roadside units (RSUs) into a centralized server, represented by the red node as shown in Figure 5-2. The red node serves as a powerful server equipped with 2 Intel Xeon Silver 4114 Processors, 1TB RAM, and V100 GPU cards with 16GB, symbolizing a cloud infrastructure. In this configuration, the red node manages all the workload of the MMOT program, including its SORT and DeepSORT algorithms. An RSU

operates on a Raspberry Pi 3 Model B, utilized solely for encoding and decoding video streams from its attached camera in this baseline. To simulate camera streams, we employed these RSUs to replay the recorded data from AIC at a speed of 10 frames per second (fps) for each camera.

Next, the setup involves adding 8 Jetson Nanos to the device-cloud setup to offload the heavy processing load of DNN-based object detections. This augmented device-cloud setup is referred to as device-edge-cloud. We utilize the Yolov5 pre-trained model for both device-cloud and device-edge-cloud configurations. A group of RSUs will be connected to a Jeton Nano via a wired network if they are geographically collocated to an intersection. When an MMOT is registered to the central server (the red node in Figure 5-2), it delegates some subtasks to the blue nodes representing Jetson Nano devices. For example, subtasks on object detection and tracking are specified in line 9 and line 10 of Listing 5. As depicted in the graph in Figure 5-2, when the number of RSUs exceeds 10, they outperform a device-cloud counterpart in terms of latency. It's important to note that, although device-edge-cloud has more processing power than device-cloud, adding one more communication hop to the network topology will introduce more delay if the red node is not overloaded in terms of processing or bandwidth. However, for the heaviest operation Yolov5, a V100 can process 100-250 fps, while each Jetson Nano can process 10-25 fps. Consequently, the benefits of having edge nodes become more apparent when there is a higher processing load and network demand, such as streaming from more than 10 cameras. This indicates the direction towards achieving the necessary performance and scalability while being able to build the system in a low-code fashion with semantic programming introduced below.

5.4 MODEL AND DESIGN

5.4.1 Semantic Programming Model for Low-code Programming

This section introduces the Semantic Programming (SP) paradigm for building and deploying applications on edge-cloud continuum, which is geared towards the integration of multimodal data. The inspiration for SP is drawn from the intricate interplay between the semantic and episodic memory systems observed in the human brain. Semantic memory represents the brain's reservoir of general world knowledge, while episodic memory is akin to a mental archive that encodes, stores, and facilitates the retrieval of personal experiences embedded in distinct spatial and temporal contexts.

The fundamental idea behind SP is to emulate these cognitive processes within programming, with the overarching goal of reducing coding complexity by leveraging semantic knowledge in tandem with human cognitive principles. Unlike conventional programming approaches that heavily rely on rigid algorithms and explicit instructions, SP encourages the design of programs that can manipulate and comprehend information based on its semantic symbols. By doing so, SP introduces a more flexible and intuitive programming paradigm that aligns with the nuanced way human cognition processes and understands data.

Our programming model not only capitalizes on these cognitive principles but also integrates them with the concept of declarative programming. In traditional programming paradigms, declarative programming involves specifying what a program should achieve rather than explicitly detailing how to achieve it. By incorporating declarative programming into SP, the paradigm seeks to create a more intuitive and human-like interaction with data. Programs are crafted to articulate the desired outcomes, allowing the underlying system to infer the most efficient means of achieving those objectives based on semantic knowledge. This departure

from rigid algorithms not only reduces coding effort but also enhances adaptability, enabling programs to dynamically respond to diverse data inputs.

In essence, SP, with its integration of declarative programming, represents a paradigm shift in the way we approach information processing. It not only mirrors the cognitive intricacies of the human brain but also introduces a more flexible and responsive programming model. This synthesis of declarative programming and semantic knowledge positions SP as a promising avenue for advancing multimodal data fusion, offering a holistic framework that aligns closely with the nuanced and associative nature of human cognition.

5.4.1.1 Semantic Program

The Semantic Program (SP) is conceived as a structured set of rules derived from semantic data. These rules are articulated using the CQELS-RL syntax, an extension derived from CQEL-QL and SHACL. The formal semantics of SSR, as outlined in [Danh21], delineate the program into two fundamental rule types: hard rules and soft rules.

Hard rules, grounded in non-monotonic common-sense and domain knowledge, serve as a bedrock of background information considered "always true." The hard rules within the Semantic Program represent strict guidelines or constraints that must be adhered to without exception. These rules contribute essential contextual knowledge to the semantic program.

In contrast, soft rules encapsulate association hypotheses, with each hypothesis assigned a weight corresponding to its probability degree. This nuanced approach enables the Semantic Program (SP) to accommodate varying degrees of certainty or likelihood within its reasoning process. By assigning weights to these hypotheses based on their respective probabilities, the SP can effectively weigh and prioritize different potential outcomes or interpretations. Essentially, soft rules provide a mechanism for the SP to incorporate probabilistic reasoning into its decision-making process, enriching its capability to handle complex semantic relationships and scenarios.

```

1 PREFIX sosa: <http://www.w3.org/ns/sosa/>
2 PREFIX smart-edge: <http://eu.smartedge.vocal/>
3 CONSTRUCT STREAM <dds://smart-node1/sp01> {
4   ?Object smart-edge:enters <:/cam/106/fov>.
5   ?Object smart-edge:isDetectedAt ?timeStamp.
6 WHERE {
7   STREAM <dds://smart-node2/datafusion01/> {
8     ?detection a smart-edge:Detection.
9     ?detection sosa:resultTime ?timeStamp.
10    ?detection sosa:hasResult ?result.
11    ?result smart-edge:hasConfidenceScore ?score.
12    ?result smart-edge:hasBox ?box.
13    ?box sosa:isSampleOf ?Object.
14    ?result smart-edge:hasDetectedObject ?DetObj.
15    ?DetObj a smart-edge:Vehicle.
16 FILTER (?score > 0.8)
17 }
18 }

```

Figure 5-3 Example of a semantic program which reasons if an object enters field of view of a camera.

In the context of UC2, the task involves computing the queue length, which requires the detection of a car entering the field of view (FOV) of a camera. The semantic program depicted in Figure 5-3 serves the purpose of updating information when a car enters the camera's FOV. Within this program, the STREAM keyword is used to indicate the source of input data and where

the results will be directed. For instance, Line 7 specifies that the input data originates from a smart node executing data stream fusion operations.

To facilitate this data exchange, the SmartEdge Runtime (as detailed in Section 5.4.2.2) must subscribe to the topic described in Line 6 to receive object detection results. The combination of the CONSTRUCT keyword and subsequent lines (3, 4, and 5) outlines how the results will be structured and returned, specifying the topic as "smart-node1/sp01/". By manipulating endpoints associated with the STREAM keyword, the orchestrator can establish links and guide the flow of data within a SmartEdge swarm.

Moreover, Line 16 introduces the FILTER operation, which can be viewed as a simple soft rule within the program. The confidence score associated with this operation serves as an adjustable parameter, allowing optimization techniques detailed in Section 5.3 to ensure program efficiency and reliable results. By dynamically adjusting the confidence score, the optimizer optimizes the program's performance, ultimately enhancing its effectiveness in processing and analyzing data.

5.4.1.2 Semantic Data Model

To establish a meaningful link between programming elements and semantic symbols that can be understood by both humans and machines, we adopt the RDF-based model of Semantic Streams. This conceptual framework utilizes RDF (Resource Description Framework) to represent sensory streams. For instance, consider a scenario where a video data stream, recorded by a camera, is treated as a sequence of symbolic observations symbolizing individual video frames. To illustrate, **Errore. L'origine riferimento non è stata trovata.** provides an example of how a frame captured from Camera 160, deployed at Junction 270 in Helsinki (UC2), is represented in RDF using the Semantic Observations, Samples, and Actuations (SOSA) ontology.

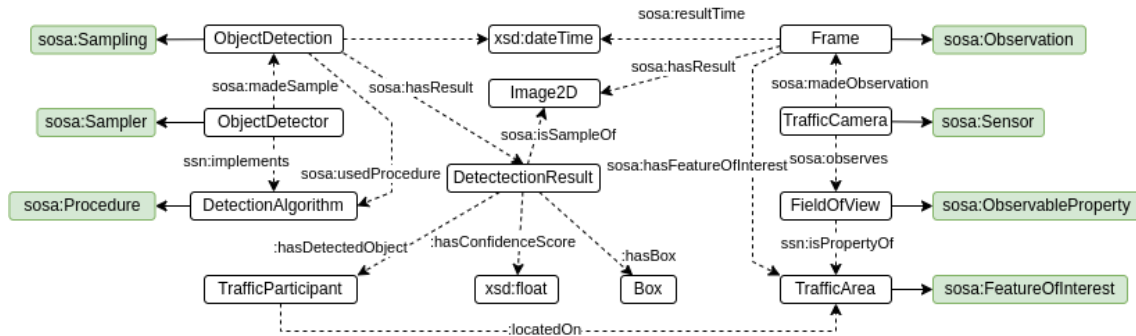


Figure 5-4. Example of data schema to describe relationships of camera, object detections and object detection results

Regarding the semantic data schema extended for SOSA ontology, illustrated in Figure 5-4, the traffic camera is conceptualized as a sensor, and the camera's field of view is modelled as its feature of interest. Specifically, the RDF representation indicates that the field of view of Camera 160 encompasses the road 270.280.Vaelimerenktu, connecting to Junction 270. These frames are then modeled as instances of the SOSA Observation class, resulting in 2D images that encapsulate the visual information captured by the camera.

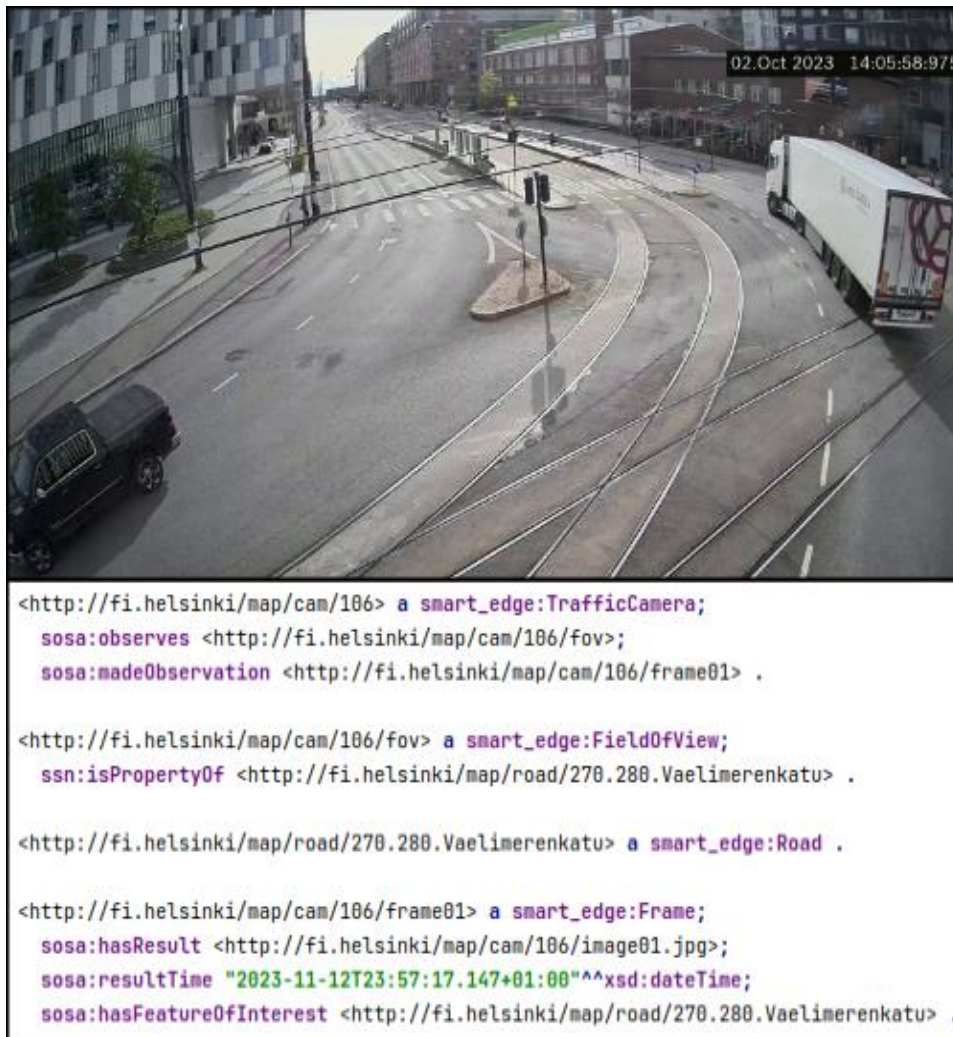


Figure 5-5. Example of a semantic stream snapshot of camera frame captured from camera 160 deployed at junction 270

The detection of a video frame is symbolized within SOSA ontology as a SOSA:Sampling. This sampling operation yields multiple detection results, essentially serving as samples derived from the 2D image (denoted as Image2D) extracted from the video frame. These detection results specifically pertain to identified objects, referred to as traffic participants, positioned within the traffic area corresponding to the camera's field of view.

The engine responsible for object detection is conceptualized as a SOSA:Sampler. This sampler, acting as an object detection engine, implements a specific detection algorithm—illustratively mentioned here as a DDN-based (Deep Neural Network-based) object detection algorithm. The SOSA ontology is used to classify and model these components within a semantic framework, enabling a clear understanding of their roles and relationships. Additionally, the temporal context of the detection process is captured through the property SOSA:resultTime, which denotes both the timestamp of the video frame and the moment at which the detection of objects within that frame occurred. This temporal information is crucial for establishing the chronological order of events and associating detected objects with specific instances in time, providing a comprehensive understanding of the spatiotemporal dynamics of the observed traffic scenario.

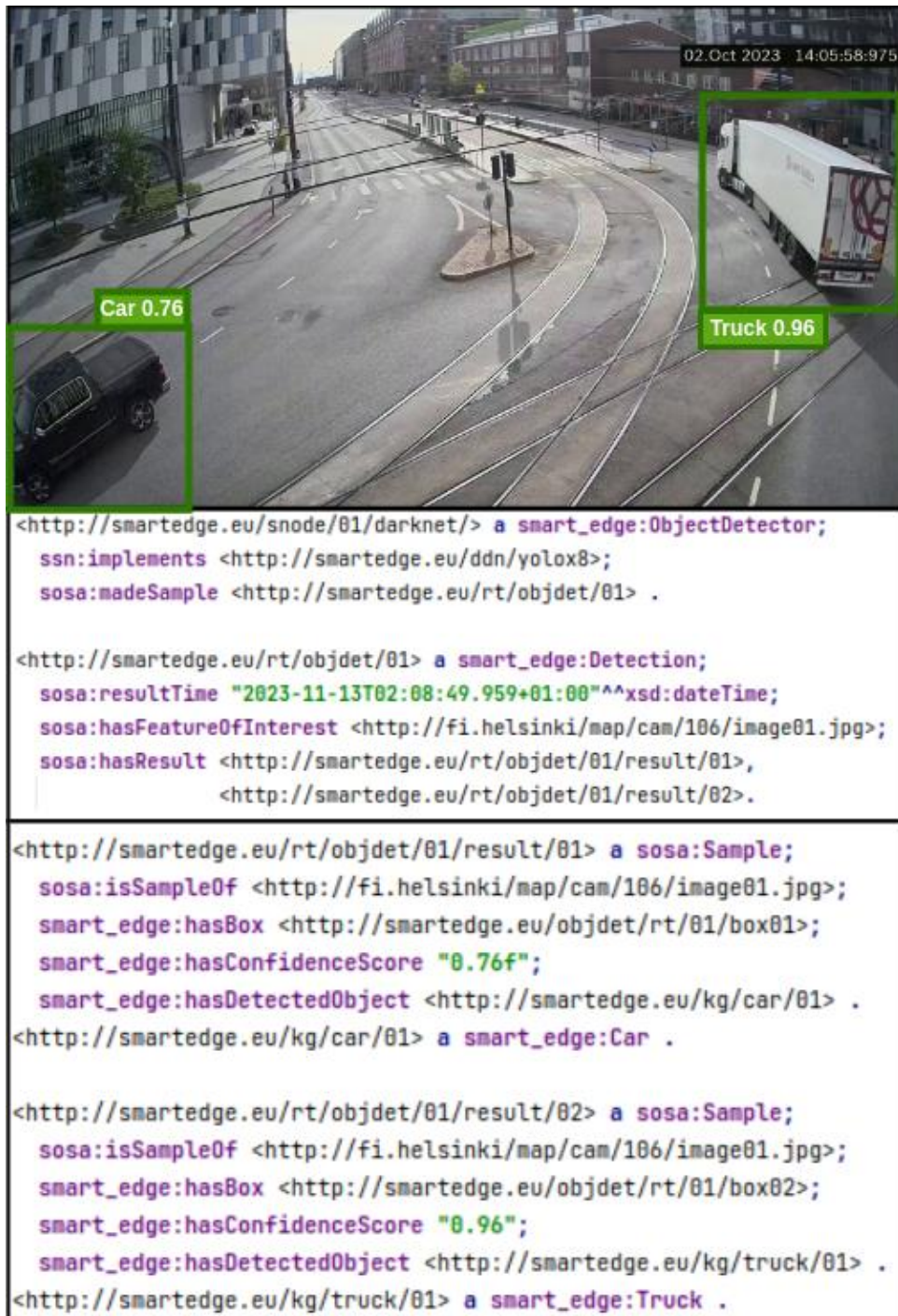


Figure 5-6. Example Semantic Streams produced by an object detection

Figure 5-6 presents a snapshot of semantic stream data, showcasing the object detection results of the frame depicted in Figure 5-5. Within this snapshot, the green boxes detected bounding boxes, indicating the presence and location of objects within the scene. The RDF representation indicates that the object detector is implemented with YoloV8[1]. The object detection of image01 yields two results: the first box represents a car with a confidence score of 0.76, and the second box represents a truck with a confidence score of 0.96.

5.4.1.2.1 Semantic data model for Traffic (UC1 and UC2)

In this domain-specific Smart Traffic ontology, the hierarchy and relations between elements are as follows: A city encompasses a road network comprising junctions and road segments. Each road segment contains one or more lanes, with each lane potentially categorized by type (such as cars, trams, buses, or truck-only lanes). Lanes may also be associated with individual junctions. Each road segment serves as a link connecting two junctions. However, each junction may function as a connecting point for two or more road segments. Lanes can be connected to other lanes at a junction.

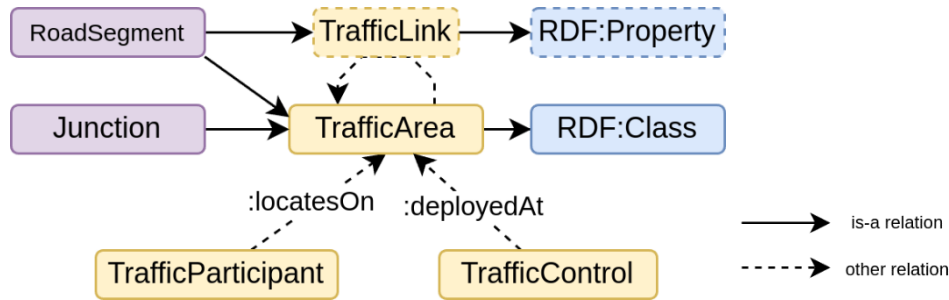


Figure 5-7. Core concepts of Smart Traffic data schema

Figure 5-7. Core concepts of Smart Traffic data schema Figure 5-7 depicts the Traffic ontology utilized in both UC1 and UC2 scenarios. In this ontology, a traffic area denotes a location where traffic activities occur. Examples of traffic areas include road segments and junctions. A traffic link serves as a connection between two traffic areas. Notably, a road segment also functions as a traffic link, facilitating the connection between two junctions. Within our model, we conceptualize the traffic link as a relational entity linking two traffic areas together.

Figure 5-8 gives a snapshot of the semantic description associated with the road segment connecting to junction 270 within the deployment site in Helsinki (UC2). The full map of the deployment site is visually presented in Figure 5-9. Junction 270 is visually highlighted by a green circle on the map, distinguishing it as a significant point within the transportation network. Lines 10 to 12 of the semantic description elucidate that Junction 300 establishes a connection with Junction 270 via the road segment identified as <http://fi.helsinki/map/roadsegment/fi.helsinki.300.270.Vaelimerenkatu>.

```

1  @prefix smartedge: <http://smartedge.eu/vocabulary/> .
2  @prefix smartedge-traffic: <http://smartedge.eu/vocabulary/traffic/> .
3
4  <http://fi.helsinki/map/roadsegment/fi.helsinki.300.270.Vaelimerenkatu> a smartedge-traffic:RoadSegment;
5  smartedge:hasName "fi.helsinki.300.270.Vaelimerenkatu" .
6
7  <http://fi.helsinki/map/junction/fi.helsinki.270> a smartedge-traffic:Junction;
8  smartedge:hasName "fi.helsinki.270" .
9
10 <http://fi.helsinki/map/junction/fi.helsinki.300> a smartedge-traffic:Junction;
11 smartedge:hasName "fi.helsinki.300";
12 <http://fi.helsinki/map/roadsegment/fi.helsinki.300.270.Vaelimerenkatu> <http://fi.helsinki/map/junction/fi.helsinki.270> .

```

Figure 5-8. . A snapshot of semantic description of a road segment connected to Junction 270 in Turtle format



```

PREFIX smartedge: <http://smartedge.eu/vocabulary/>
PREFIX smartedge_traffic: <http://smartedge.eu/vocabulary/traffic>

SELECT ?roadSegment WHERE {
  ?J1 a smartedge_traffic:Junction;
      smartedge:hasName "fi.helsinki.270".
  ?J2 a smartedge_traffic:Junction.
      smartedge:hasName "fi.helsinki.267".
  ?J1 ?roadSegment+ ?J2.
  ?roadSegment a smartedge_traffic:RoadSegment;
}

```

Figure 5-9. SPARQL query to find all road segments connect two Junction using property path.

With our sophisticated data model, we possess the capability to identify and extract all the links connecting two distinct traffic areas on a map. As an example, consider the top portion of Figure 5-9, which delineates the road network infrastructure of the deployment site earmarked for the UC2 implementation in Helsinki. Utilizing our data model's robust features, we can employ a SPARQL query to systematically list all the road segments that establish a connection between Junction 267 and Junction 270. This capability is made possible by leveraging the properties path feature provided by SPARQL, enabling us to navigate and query complex network structures with ease and precision.

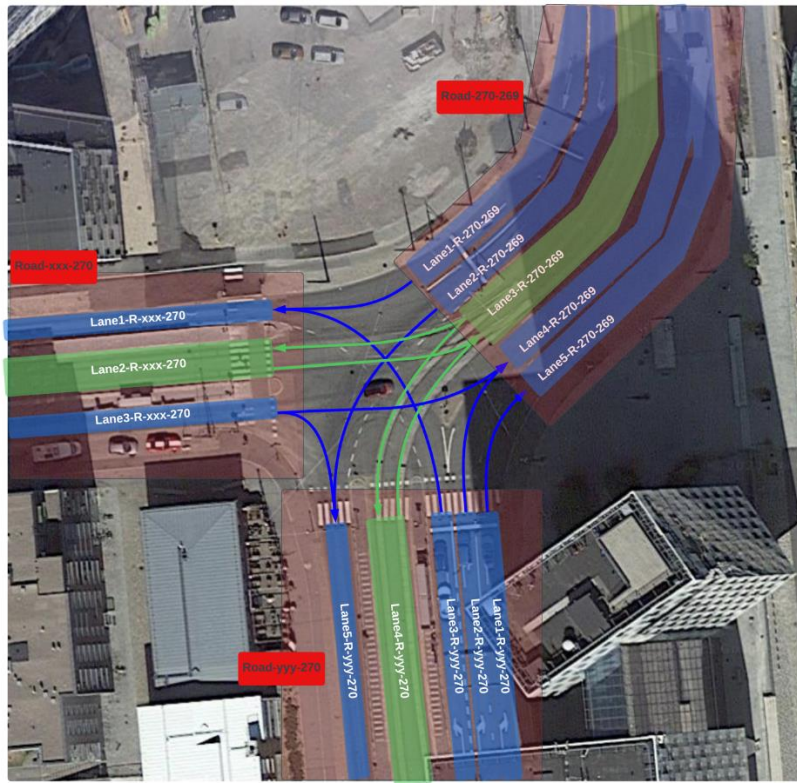


Figure 5-10. Lane layout at Junction 270 in Helsinki serves as an example.

Before presenting the relationship between Lanes, Road Segments, and Junctions, let us introduce the lanes outlined within a junction. For example, Junction 270, as shown in Figure 5-10. A road segment typically comprises multiple lanes, each designated for specific types of traffic participation, such as vehicles (blue lanes) or trams (green lanes). Each lane originates at a junction and terminates at the next junction. Arrows connecting lanes represent connections between two lanes crossing a junction. These relationships are illustrated in Figure 5-11.

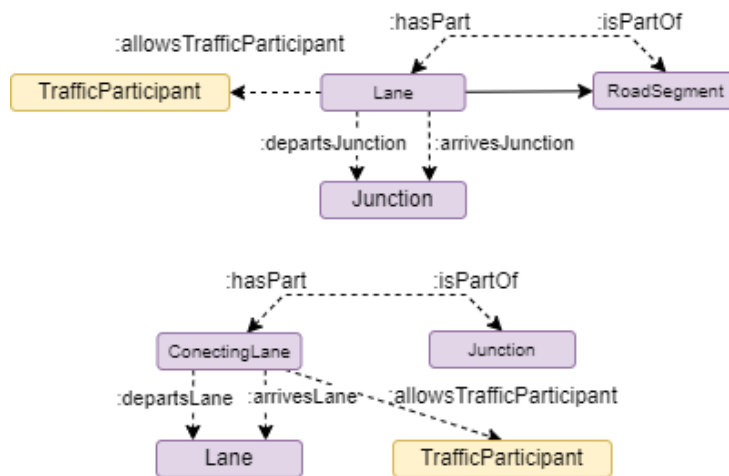


Figure 5-11. The relationships between Lane, Junction and Connecting Lane

5.4.1.2.2 Semantic SLAM (UC 3 and UC4)

Next, we introduce another form of Semantic Streams, Semantic SLAM (Simultaneous Localization and Mapping). In UC2 and UC4, SLAM is a technique that allows robots to create and update a map of an unknown environment while estimating their own location within it. SLAM is essential for many applications that require autonomous navigation. The problems and challenges associated with it have been described in UC3 for the robots on factory floors. However, SLAM poses several challenges, such as dealing with noisy and incomplete sensor data, maintaining consistency and accuracy of the map, and scaling to large and dynamic environments.

One way to address these challenges is to use semantic information to enrich the SLAM process via Semantic SLAM. Semantic information refers to the meaning and context of the entities and relations in the environment, such as objects, places, events, or actions. Semantic SLAM can help robots to identify and classify the elements of the map, to reason about their properties and behaviors, and to communicate and cooperate with other robots or humans.

5.4.1.2.2.1 RDFize ROS-based SLAM data structures into Semantic Streams

To transform ROS-based SLAM data structure of SmartEdge's Semantic Data model, we first map 2D/3D map data elements in C-structure into corresponding RDF types.

- **2D SLAM Maps.** In ROS2, Occupancy Grid is used to represent a 2D occupancy grid map. The map is represented as a 2D array of integers, where each integer represents the probability that a cell is occupied by an obstacle.
- **3D SLAM Maps.** OctoMap [Hornung13] is used to represent a 3D occupancy grid map. The map is represented as a 3D voxel grid, where each voxel is either occupied or unoccupied. In addition to OctoMap, Point Cloud is another way to represent a 3D SLAM map. A point cloud is a collection of points in space, each of which has a position and a color. Both OctoMap and Point Cloud are widely used data structures for representing 3D SLAM maps.

To RDFize the SLAM map and elements, we first need to define the RDF ontology by determine the key concepts and entities within SLAM map data, such as maps, landmarks, robot poses, and occupancy information, then create RDF classes and properties based on ROS data format. The Figure 5-12 is samples ontology that represent the data type of 2D SLAM map (Occupancy Grid) and its components.

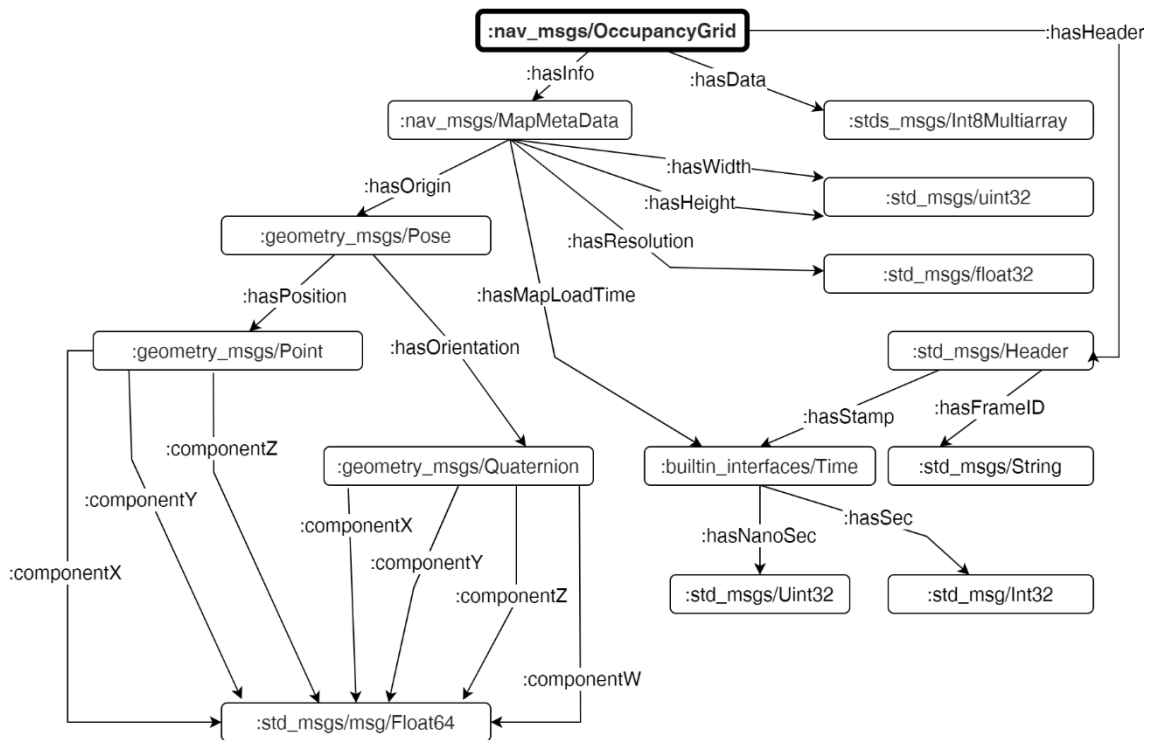


Figure 5-12. Occupancy Grid represents a 2-D grid map.

Once we have the ontology, we could start mapping the SLAM data to RDF. It requires to create a mapping between the fields of SLAM data (e.g., grid cells, landmarks, poses) and corresponding RDF classes and properties defined in the ontology. After that, the data could be transformed to RDF triples format for representation, storage, and querying using Semantic Web technologies.

By RDFizing the ROS-based SLAM data structure, we can make it interoperable with other RDF data sources and enable semantic queries and reasoning over it.

5.4.1.2.2.2 Link Semantic Types of Spatial Objects

Spatial objects are physical entities existing in the robot's environment that the robot interacts with or perceives. Spatial objects can be used to describe and classify the elements of a SLAM map, such as objects, places, landmarks, or trajectories. These objects can have various shapes, sizes, and materials. Understanding and modeling spatial objects accurately is crucial for robots to perform tasks effectively and safely in real-world environments. This involves techniques such as sensor fusion, simultaneous localization and mapping (SLAM), object recognition, and scene understanding in this SmartEdge project.

In this work, we depict spatial objects within the semantic map using a 2D SLAM map, also known as occupancy grid maps, along with their components. A semantic map allows us to represent and manipulate various objects in the environment, such as furniture, doors, or obstacles. To link the semantic types of spatial objects to the map, we assign semantic labels to the cells of the occupancy map, in that way, we can identify and classify the objects, and query their properties and relations. Overall, a semantic map can provide a high-level and abstract view of the environment and facilitate the understanding and communication of the robot and the human.

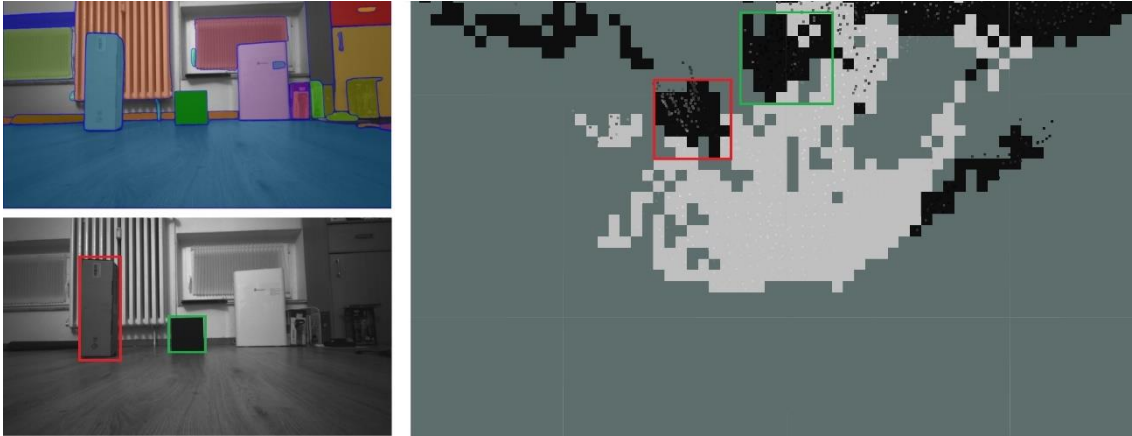


Figure 5-13. Semantic SLAM. Camera stream (left) and Occupancy Grid map (right) with object detection

Figure 5-13 showcases a robot constructing a Semantic SLAM map using an RGB-D camera. On the left side of the figure, there's a live video feed from the camera showing the environment clearly. On the right side, there's a map called the Occupancy Grid Map, created using depth sensor data and IMU data. Each square cell in the map shows if that space is free (white), blocked by obstacles (black), or unexplored (dark green).

The camera's data is crucial for the system, serving as its main source of information. It is used to analyze the data to get the semantic meaning for the robot to understand the environment better. The camera feed is processed by a perception module that performs object detection (Figure 5-13 bottom left) using ML models mentioned in section 5.3.1 such as YOLO [Redmon15], DETR [Carion20] and performs segmentation (Figure 5-13 upper left) using ML models such as Segment Anything Model [Kirillov23], Grounded SAM [Ren24] to get the semantic category of the spatial objects and figure out the location of the objects in the environment. Then, this information is used to update the Occupancy Grid Map. This creates semantic meaning for the map, giving the robot a deeper understanding of its surroundings. With this knowledge, the robot can navigate smarter, make better decisions, and interact with its environment more effectively.

5.4.1.2.2.3 Continuous Queries over Semantic SLAM

To query and manipulate objects within the semantic map, we need to use a query language that can access and manipulate RDF data, such as SPARQL, a standard for querying and updating RDF graphs. SPARQL allows us to express patterns of RDF triples, and to apply filters, operators, and functions to them.

A continuous query is a query that is executed repeatedly over a stream of data and produces a stream of results that reflect the changes in the data. In this use case, the robot continuously scans the environment and updates the map continuously, so that the robot needs to get the latest information to make a decision for tasks such as path planning and navigation.

Consider a robot that is navigating a cluttered warehouse. The robot has an RGB-D camera that it uses to scan the environment and generate a probability occupancy grid. The robot starts at a known location and scans the environment in front of it. As it scans, it updates the probability occupancy grid, assigning higher probability values to cells that are likely to be occupied and lower probability values to cells that are likely to be free. During the mapping process, in conjunction with spatial objects information, the system constructs the semantic map using the techniques described above, making it ready for querying and manipulation. For example, the

Figure 5-14 show an idea query **Errore. L'origine riferimento non è stata trovata.** that find all the free cells from the Semantic SLAM map that will be used as the input for path planning tasks.

```
SELECT ?cellX ?cellY
WHERE {
  ?cell a :nav_msgs/OccupancyGrid.
  ?cell :hasData ?data.

  ?data :hasCells ?cell.
  ?cell :value ?probability.
  ?cell :componentX ?cellX .
  ?cell :componentY ?cellY.

  FILTER (?probability <0.3)
}
```

Figure 5-14. Sample SPARQL query for path planing

5.4.2 Integrate/deploy/build toolchain into execution target/environments.

5.4.2.1 Ontology-driven Design for Cross-Layer Toolchain

Figure 5-15 illustrates the overview of designed components of SmartEdge toolchain. Application specification is compiled into semantic program which is sent to the SmartEdge orchestrator and optimiser developed in T5.3. The core of the toolchain is the SmartEdge Runtime executes primitive processing operators to serve the domain-specific applications. The SmartEdge processing primitives include two types operators, namely Graph Stream Query Operators (cf Section 5.4.3.1) and Tensor Computations (cf. Section 5.4.3.2). These operators provide the core primitives for Multimodal Stream Fusion operations developed in T5.1 (cf. Section 2). To implement such operations, there are several software components needed to be integrated and abstracted as described in Section 5.4.2.

The Network Ops are facilitated by SmartEdge network components developed in WP4 (c.f D4.1). Based on such Network Ops, a set of Plugins and a set of Connectors are provided in Section 5.4.4 and Section 5.4.5 respectively. These Plugins and Connectors help lower the effort of UC developers in integrating lower-layer components as well as wiring different components to domain-specific application logics.

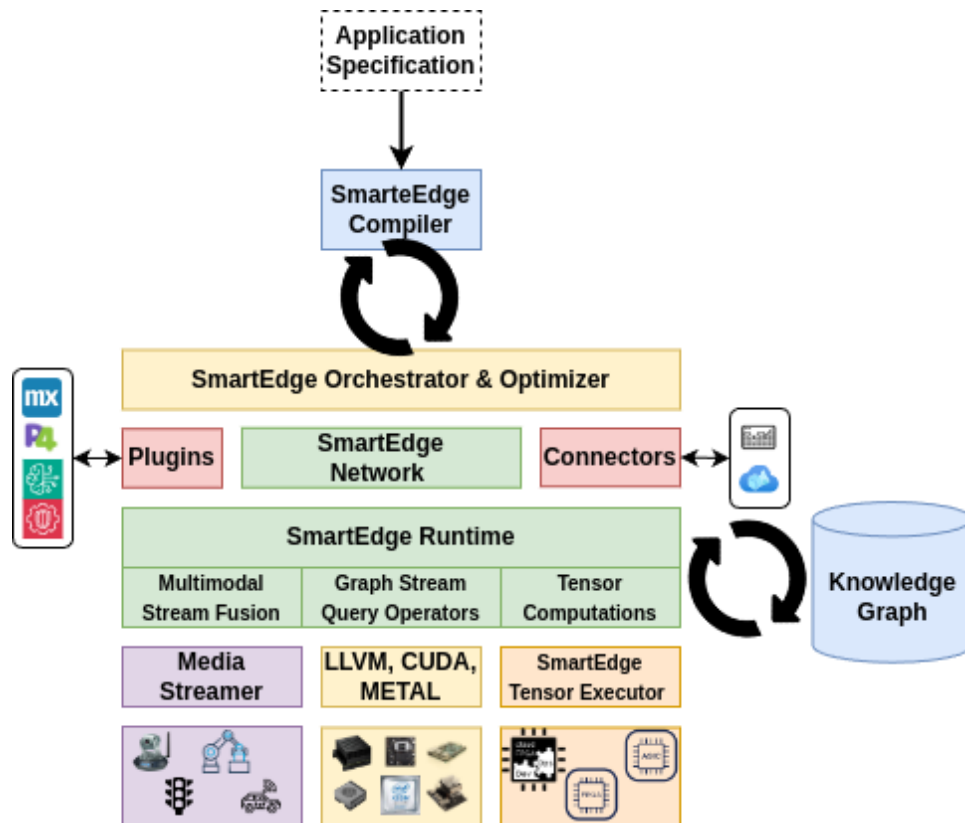


Figure 5-15. Overview of component design of SmartEdge low code toolchain

The first step in building such a toolchain is extending the schema defined in WP3 to accommodate vocabularies needed to model attributes and features (e.g. connecte sensor, CPU/GPU, host OS) of runtime deployment context under an ontology, called Execution Context Ontology (ECO). Then, ECO must be extended further to link with technical aspects (e.g. network interfaces, targeted P4 runtimes) of WP4 as well as components provided for it. Also, semantic description of data fusion to be developed in T5.1 will be incorporated in runtime toolchain configuration. With provided vocabularies, T5.4 will be working with UC partners to populate instances based on concepts and attributes to expand ECO further in D5.2 and D5.3 as the implementations of UCs progress until the end of SmartEdge or even beyond. For example, all desired configurations towards the expected demonstrations in Helsinki will be described and instantiated to compile small-scale lab deployments with similar hardware at TUB to do performance profiling and tuning to choose the best potential artifacts before doing field tests in Helsinki. A similar process can be built to run on robotic simulations in UC3 in development phase, which can then be implemented in field tests in the Dell manufacturing lab.

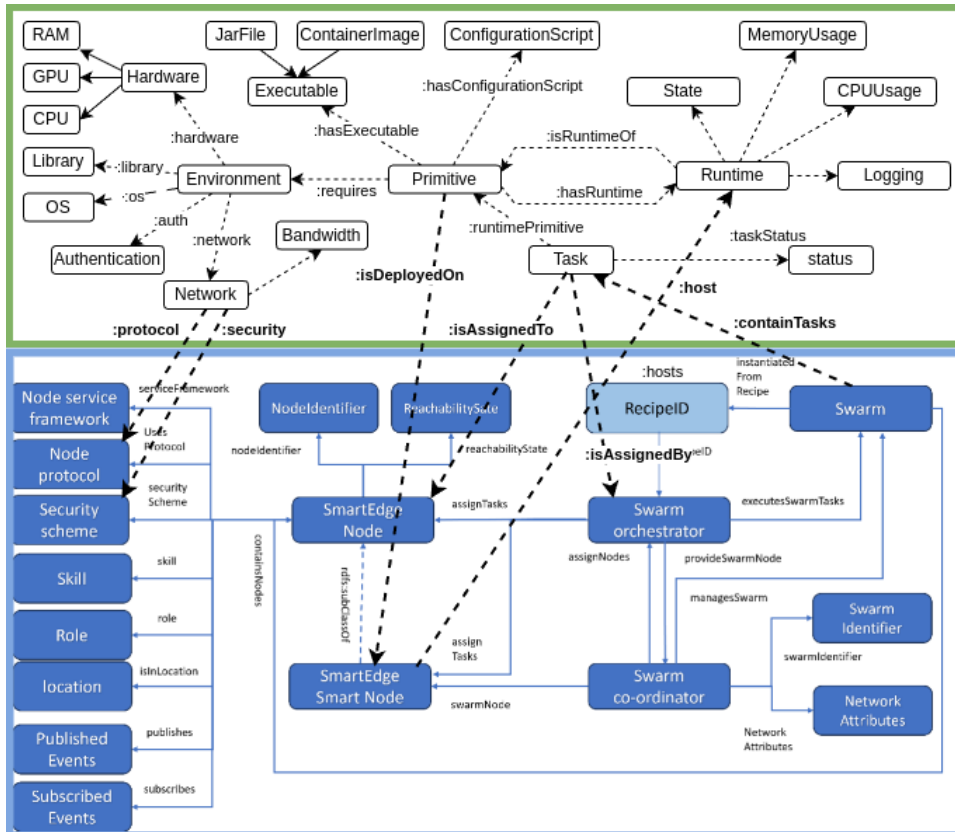


Figure 5-16. Execution Context Ontology extended from SmartEdge Schema from D3.1

5.4.2.2 SmartEdge Runtime

The SmartEdge Runtime (SRt) serves as a critical component hosted within the infrastructure of SmartEdge smart-nodes. Its primary function is to provide a suite of advanced functionalities that empower a smart-node to effectively contribute to a larger swarm network. Specifically, SRt enables the execution of semantic programs by managing and executing the runtime of primitive operations declared within these programs. These operations encompass a wide range of tasks, including data fusion, stream query operations, and tensor computations specified in Section 2 and 5.4.3.

One of the key strengths of SRt lies in its seamless integration capabilities. It effortlessly interfaces with a variety of companion processes, including companion runtimes such as P4 Runtime, and plugins designed to enhance the network’s functionalities. Additionally, SR is designed to interface with remote processes via protocols like DDS and UPC-UA, further extending its reach and interoperability.

The processing state and configuration details are meticulously tracked and maintained within SRt’s Distributed Knowledge Graph (DKG). This centralized repository ensures that the system remains coherent and can be efficiently managed. Moreover, SR leverages peer-to-peer (P2P) data querying and federation mechanisms, enabling nodes to cross-reference and query states stored within the DKGs of their peer nodes. This capability enhances the system’s resilience and scalability, facilitating seamless communication and coordination across the swarm network.

Figure 5-17 delineates the architectural blueprint of the SmartEdge Runtime (SR), elucidating its fundamental components and their specialized functions. Positioned atop SR is the Message Manager, acting as a pivotal ingress point for control messages and request messages originating

from disparate system components. These messages encompass a gamut of communication types, including state request messages aimed at soliciting real-time status updates from SR or dispatching operational directives.

At the heart of SR lies the State Manager, orchestrating the dynamic upkeep of a smart node's operational state. This entails disseminating critical node availability data and prognosticating state transitions, such as the potential disengagement of a smart-node from the swarm in response to spatial reconfigurations. Simultaneously, the Hardware Manager assumes responsibility for overseeing the entire hardware inventory within the smart node's purview. During the bootstrapping phase, it meticulously scans hardware specifications, transmuting them into RDF format, and archives them within the Distributed Knowledge Graph (DKG) for subsequent reference and management.

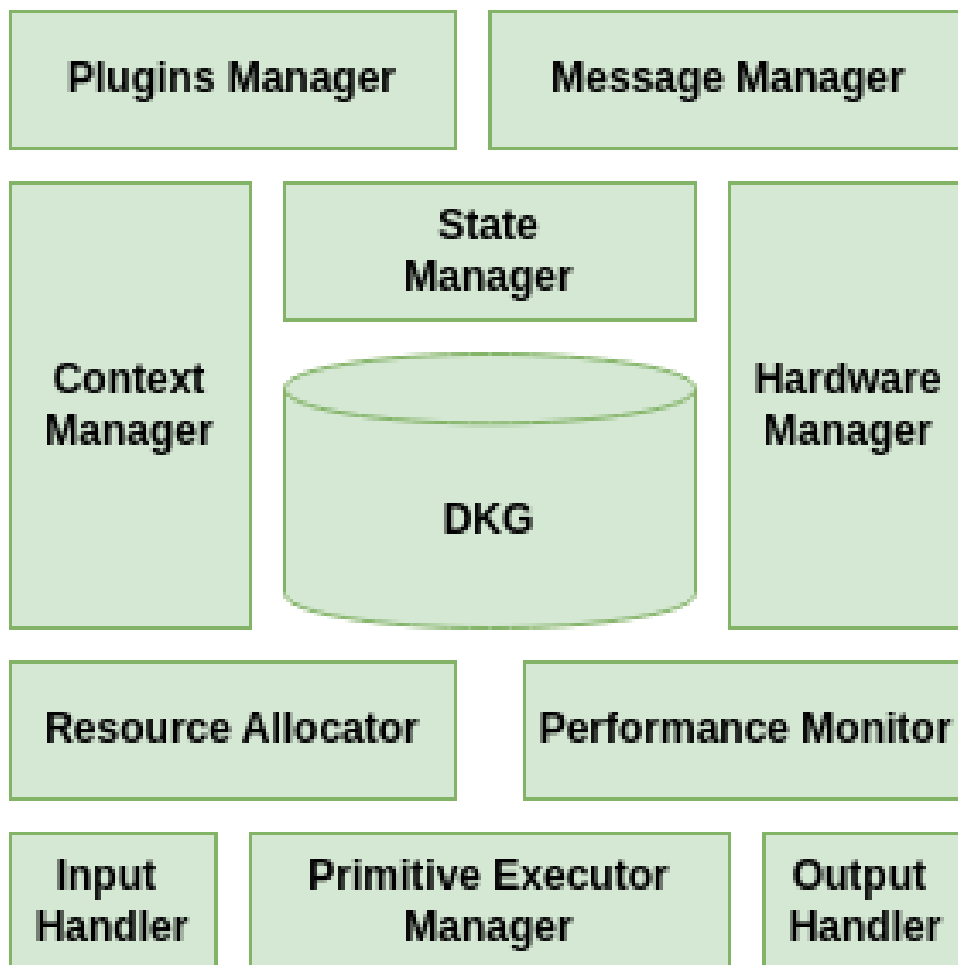


Figure 5-17. Component design of SmartEdge Runtime

The Context Manager is tasked with curating and managing environmental insights pertinent to the node's operational milieu. It interfaces seamlessly with deployed sensor arrays, assimilating and updating sensor-derived data within the Distributed Knowledge Graph (DKG). This data serves as a linchpin for predictive analytics within the system, enabling the State Manager to forecast node states with heightened accuracy.

The Primitive Runtime (PRT) Manager assumes a pivotal role in the SR ecosystem, overseeing the deployment, execution, and maintenance of primitive executables. Operating in tandem, the Primitive Executor leverages semantically annotated executable requirements sourced from

the DKG, creating bespoke virtual environments tailored to each execution instance. These environments ensure consistent runtime execution conditions, facilitating robust and reproducible outcomes across successive executions.

Simultaneously, the Input Handler serves as a conduit for managing data ingress to fuel primitive operations. It discerns and employs appropriate communication protocols to interface with diverse data sources, ensuring seamless data acquisition. Conversely, the Output Handler marshals computation results emanating from primitive executions, furnishing actionable insights for downstream processing.

Augmenting SR's capabilities are the Resource Allocator and Performance Monitor, interfacing seamlessly with the Primitive Runtime Manager. The Resource Allocator orchestrates the judicious allocation of computational resources, optimizing parameters such as RAM allocation and CPU core provisioning. Concurrently, the Performance Monitor meticulously records runtime performance metrics, offering granular insights into operational efficiencies. These components synergize with the Optimizer module to dynamically recalibrate resource allocation strategies in response to evolving performance metrics, ensuring optimal operational efficacy within the SR ecosystem.

A Runtime can be packaged in several distinct configurations to cater to varying deployment requirements:

- **Lightweight Version:** For simple or resource-constrained environments, a lightweight version of the Runtime can be packaged to run within a Java Virtual Machine (JVM). This version offers the advantage of easy deployment and compatibility across different platforms, leveraging the JVM's portability.
- **Complex Version with Companion Runtimes or Plugins:** In scenarios demanding advanced functionalities or intricate integrations, a more sophisticated version of the Runtime can be deployed within a Dockerized environment. This approach enables encapsulation of the Runtime along with its companion runtimes or plugins within Docker containers, ensuring isolation and portability across diverse computing environments.

Within the Dockerized environment, a Knowledge Graph (KG) tailored for the tool chain facilitates the packaging process. This KG contains metadata pertaining to host environments, encompassing hardware specifications, sensor configurations, input/output semantics, and network capabilities. By leveraging this metadata, the packaging process is streamlined, with the KG guiding the generation of Docker Compose configurations semi-automatically. This semi-automation minimizes manual intervention and ensures consistency in the deployment process, enhancing efficiency and reducing deployment overhead.

5.4.2.3 Interactive Active Model training and selection workflow

In this section, we aim to develop an automated low-code training platform, tailored for domain experts, that focuses on learning on edge devices. Currently, edge devices can help to collect vast amounts of data. However, training directly with this vast data is slow on edge devices, and those with lower computational power may not support effective training. Therefore, we plan to tackle this challenge from two perspectives. First, utilizing active learning, we aim to allow domain experts to select the most representative data points in each training loop for model learning. Building on our previous project, VisionKG, we aim to enrich the training corpus by querying for features closely matching those annotated by domain experts, thereby enhancing

the training set. To improve generalization ability, we will extract the foundational model from pre-trained model zoo and leverage transfer learning effects. Then, we will use an interactive model selection strategy to choose and optimize models based on users' or domain experts' specific requirements. Self-adaptive and data-centric learning are pivotal in robust computer vision, and crucial for scenarios like autonomous vehicles and surveillance. Implementing robust learning pipelines across conditions such as adverse weather and varying illumination is challenging but essential for deploying trained models. Hence, our goal is to develop a robust, user-friendly low-code training platform for real-world applications, enabling the selection of the most representative data/model based on domain knowledge from target users. We will begin with VisionKG, which can organize, manage, and access visual datasets, and automate training/evaluation pipelines for computer vision applications using knowledge graphs and Semantic Web technologies. A unified schema in VisionKG enables efficient data retrieval and querying using both SPARQL and natural language, facilitated by large language models. Drawing on [Anh21, Kien21, Jicheng23], we will enhance its capabilities for better integration with active learning, focusing on model training and selection through advanced semantic analysis, continuous learning, and model adaptation. This effort seeks to create robust visual recognition systems, enhance semantic interpretations, and support an efficient, interactive model training and selection in a data-centric approach.

Initially, VisionKG follows Linked Data principles and FAIR data guidelines to enhance the findability, accessibility, interoperability, and reuse of the integrated datasets. It offers three main capabilities to improve model training workflows:

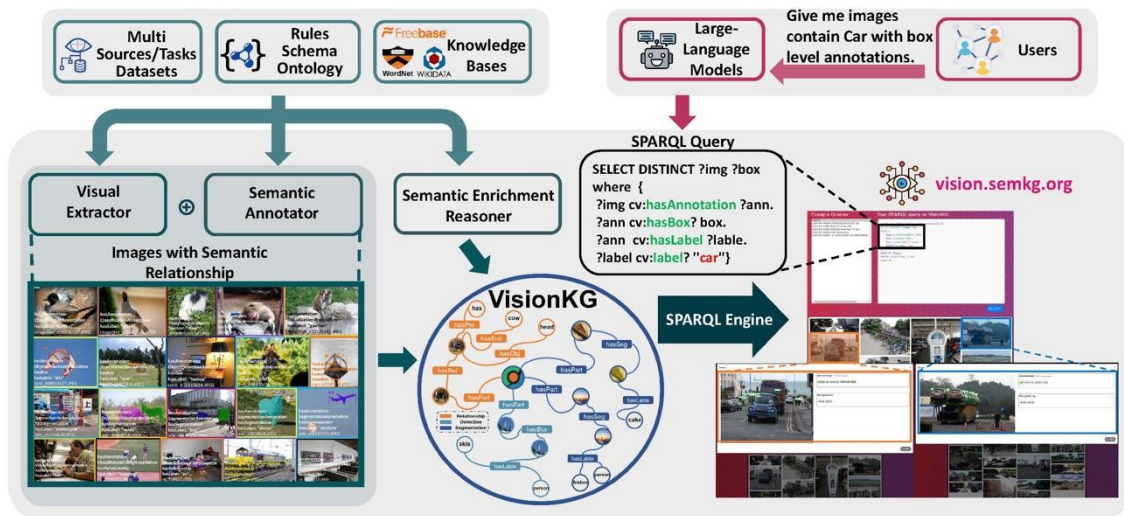


Figure 5-18 Overview of VisionKG Platform

- Unified access visual datasets integrated with active learning loop: Users can query across datasets using SPARQL to retrieve images and their corresponding annotations meeting specified criteria, aligned taxonomies via knowledge graph in VisionKG enable users to curate composite datasets easily.
- Automating robust incremental learning pipelines towards computer vision: With VisionKG, users can automate data retrieval, preprocessing, model training, evaluation etc. using integrated Python API and existing model zoo. Employing active learning and incremental learning, VisionKG can ensure robustness while preserving acquired knowledge.

- Active model updating and selection via advanced semantics fusion: Diverse data from VisionKG helps improve model robustness and reduces overfitting to specific scenarios. Users can create composite datasets as required to enhance robust evaluation under varying conditions, such as weather, illumination, and contexts. Benefiting from active learning and feature diversity in VisionKG, even with limited data points in the target domain, users can effectively transfer learning towards specific user scenarios.

To access datasets via queries, VisionKG offers a SPARQL endpoint and Python API, allowing queries based on tasks, datasets, categories, licenses, etc. An interactive GUI enables users to construct queries and visually retrieve images. For instance, as demonstrated in Figure 5-18, users can retrieve images from various sources, including categories such as person, pedestrian, and man, all aligned within the visual knowledge graph. Thus, VisionKG's consistent taxonomy simplifies querying visual data at various specificity levels, easing the combination of heterogeneous datasets. Starting with a unified data schema and integrated rich data, VisionKG will incorporate active learning and a composite data selection mechanism, both visual and semantic. This design aims to identify and prioritize crucial training data samples, focusing on uncertainty and diversity. After each training round, a feedback mechanism will be introduced, allowing iterative refinement of data selection criteria and the training loop, aligned with learning objectives or user needs. Its goal is to accelerate model learning and introduce human-in-the-loop active learning, positioning VisionKG as a leader in data-driven model training, even under sparse data or annotation conditions. Based on VisionKG, to realize interactive active model training, labeling work is performed on the limited target data with the help of knowledge from domain experts. According to the annotations, the nearest features are retrieved via SPARQL from built-in feature store in VisionKG as a supplement in the target dataset and serve to a jointly trained loop. In addition, in each training loop, users can decide whether to increase the annotated data size in the target domain according to the real-time performance of the model or remove queried data points from VisionKG, thus realizing the requirements in specific scenarios with a better generalization ability.

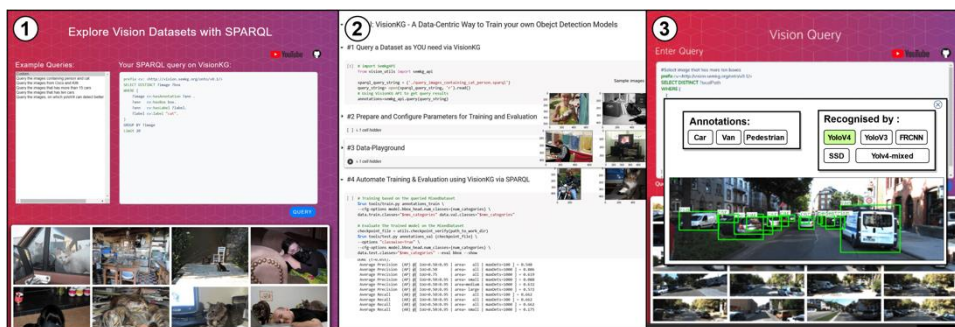


Figure 5-19 Data Querying and Pipeline Constructing via VisionKG

Additionally, VisionKG allows automating training data retrieval and model building by integrating with ML frameworks like PyTorch, as shown in Figure 5-19. For example, a simple SPARQL query can retrieve annotated images of cars and persons from various sources. These can then be fed into an object detection model, such as FRCNN [Shaoqing15], configured with a desired framework. This automated workflow accelerates experimentation by avoiding repetitive data-wrangling tasks. Besides, the diversity of datasets in VisionKG helps reduce bias and overfitting compared to models trained on a single dataset. Users can leverage additional metadata, as shown in Figure 5-20, such as weather, illumination condition, as well as image resolution, etc. to create composite datasets with enriched semantic and visual features. For

instance, users can query cars in rainy nighttime conditions across datasets. Hence, training models on such adversarial instances benefit from enhancing the model's robustness to varied real-world conditions, thus improving model performance on rare-seen scenarios. In this project, our roadmap includes developing a dynamic system for active model updating and selection. A critical feature is the feedback loop, where continuous monitoring of model performance guides updates benefiting from the data diversity in VisionKG and domain knowledge from experts, it will persistently update models with fruitful data, employing active learning and incremental learning to ensure adaptation while preserving acquired knowledge, thus keeping the model effectiveness in specific use scenarios and meeting desired requirements towards diverse visual tasks.

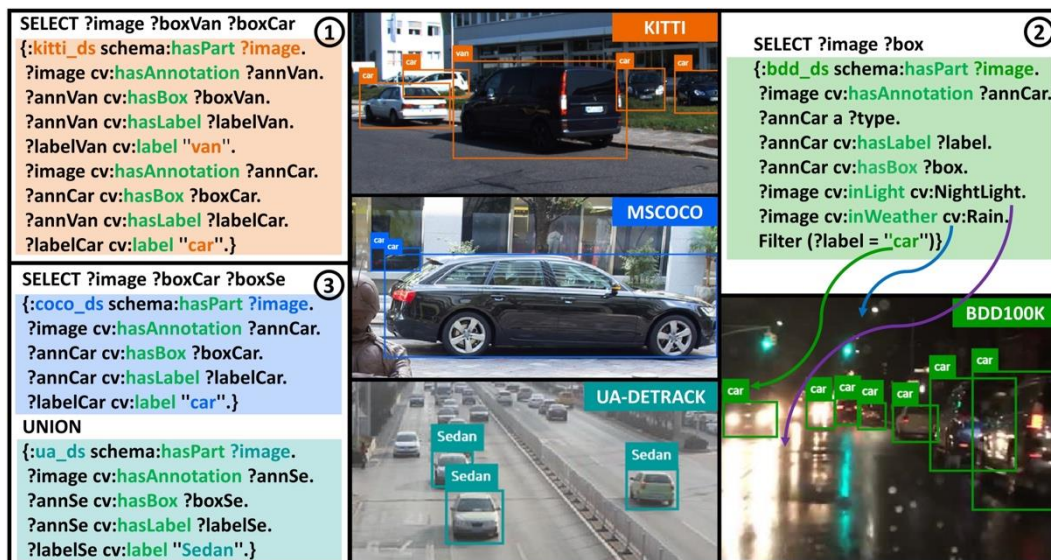


Figure 5-20 Data Diversity in VisionKG

To date, VisionKG includes 617 million RDF triples describing approximately 61 million entities from 37 datasets and four popular computer vision tasks, as shown in Figure 5-21. It aims to optimize the efficiency and effectiveness of the learning process and constitute a low code end2end platform in machine learning towards computer vision tasks, leading to ease the learning curve of users and accelerate the robust training pipeline in one stop. Besides, VisionKG is easily extensible and will empower communities to grow around the provided resources (unified data schema and fruitful pretrained models) and can serve as a blueprint for many digital data resources as the functionalities provided are generic and reusable. Furthermore, benefiting from interactive active model training and selection workflow in VisionKG, users can inject their own domain knowledge to effectively train models on limited data and edge devices with low-computing-power according to specific usage scenarios.

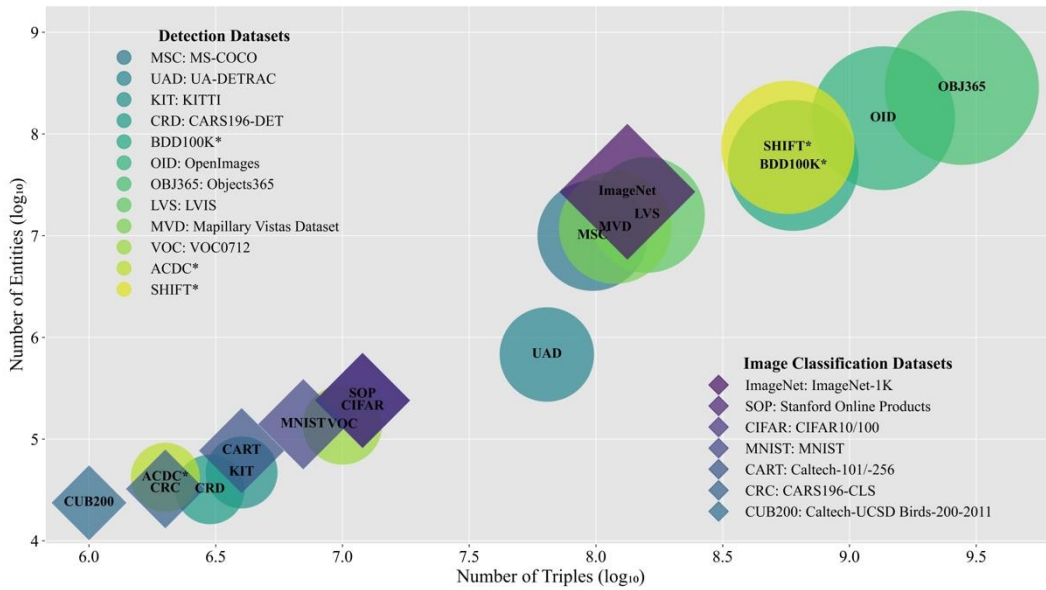


Figure 5-21 Statistics of Triples and Entities in VisionKG for Object Detection and Image Classification Datasets

5.4.2.4 Abstracting hardware and execution framework for MLOps

The design of SmartEdge Runtime necessitates the utilization of heterogeneous hardware architectures to address the diverse computational requirements of distributed applications. In addition to the challenges posed by heterogeneous hardware programming, its Runtime introduces complexities associated with managing streaming data and sensor fusion. The real-time nature of edge applications, coupled with the proliferation of sensor data streams, accentuates the need for efficient data processing and fusion techniques. Moreover, the dynamic nature of edge environments mandates the ability to seamlessly add or remove hardware resources at runtime, further complicating the design and optimization of edge computing systems. This section explores the necessity of heterogeneous hardware programming for edge intelligence via analysing state-of-the-art approaches, and evaluates prominent frameworks like OpenCL, TornadoVM, CUDA and oneAPI SYCL DPC++.

Edge computing environments of SmartEdge UCs are characterized by their distributed nature and proximity to data sources. This proximity offers reduced latency and bandwidth requirements, making it ideal for real-time data processing and analysis. However, the diverse nature of edge applications demands a heterogeneous mix of hardware components. From CPUs and GPUs to FPGAs and ASICs, each hardware type offers unique advantages suited for specific tasks. Hence, programming such applications poses significant challenges. Traditional software development models struggle to exploit the full potential of diverse architectures efficiently. Managing data movement, optimizing performance across different devices, and ensuring seamless integration are among the primary challenges. State-of-the-art approaches leverage parallelism, offloading computation to specialized accelerators, and abstracting hardware complexities through high-level programming models. Types of hardware are considered in SmartEdge toolchain:

- Central Processing Units (CPUs): Versatile and suitable for general-purpose computing tasks. CPUs offer high single-threaded performance but may lack the parallel processing capabilities required for certain edge applications.
- Graphics Processing Units (GPUs): Excel in parallel processing tasks, making them ideal for tasks like image and video processing. However, they consume more power compared to CPUs which necessitates careful consideration in energy-constrained edge environments.
- Field-Programmable Gate Arrays (FPGAs): Provide flexibility through reconfigurability, making them suitable for diverse edge applications. FPGAs offer low latency and energy-efficient acceleration but require specialized expertise for programming. With low latency and energy-efficient acceleration capabilities, FPGAs are particularly preferred for tasks demanding real-time processing and optimization.
- Application-Specific Integrated Circuits (ASICs): Custom-built for specific tasks, ASICs offer unparalleled performance and power efficiency but lack flexibility and adaptability. However, their lack of flexibility and adaptability may limit their applicability in dynamic edge computing environments.

Since CUDA and oneAPI are frameworks mainly designed to work specifically with NVIDIA devices, it would impose limitations on the possibilities of device choice. T5.4 considers open standard based frameworks such as OpenCL and TornadoVM which is a high-level tool built over OpenCL.

- OpenCL: OpenCL (Open Computing Language) is an open standard for parallel programming across heterogeneous platforms, enabling developers to harness the computational power of CPUs, GPUs, and other accelerators [Munshi09]. It uses the CPU-based host device to manage the application run on a target device. OpenCL offers fine-grained control over hardware computational resources and memory management. It requires expertise from the developer to be able to construct the application by following OpenCL development paradigms and to handle all intricacies stemming from a heterogeneous development environment. OpenCL offloads compiled kernels written in the OpenCL C language that execute on the target devices. It is a widely accepted industry standard that is used in various fields such as scientific computing, image and video processing, ML and AI, etc. OpenCL enables the execution of programs on heterogeneous hardware environments through its runtime system and execution model. It offers platform and device discovery to enable developers to optimally configure the application execution flow. Additionally, it enables task and data parallelism, memory management and data transfer, and dynamic load balancing and resource utilization.
- TornadoVM: TornadoVM Framework facilitates the rapid development of edge applications by abstracting hardware heterogeneity through a high-level programming model [Fumero19]. It offers support for various hardware accelerators and simplifies deployment across heterogeneous environments. It can reconfigure applications at runtime for hardware acceleration by utilizing the available hardware resources. While Tornado streamlines the development process, its higher level of abstraction may entail trade-offs in terms of performance optimization and fine-grained control over hardware resources. It is a unified framework written in Java and targeting parallel processing on CPUs, GPUs, and FPGAs by following the rule, write once, run anywhere. It offers seamless integration of existing software by introducing minimal changes to the source code to adapt the application to TornadoVM API paradigms. Developers can utilize its

API to create tasks to be run on target hardware and design the code flow on a high level. In the background DistributedMLOps analyses the code, data dependencies between tasks, code regions that can be parallelized, translates high-level Java into low-level OpenCL kernels, data offloading to and from the devices, and explores optimal hardware configuration to optimize the set KPIs. Tornado VM is integrated within Java VM and it utilizes the bytecode generator to perform JIT (just-in-time) compilation of Java kernels into OpenCL ones with the target vice platform in mind.

5.4.2.4.1 Profiling and Analysing TornadoVM and OpenCL

Despite providing high-level abstraction for heterogeneous hardware programming, TornadoVM quickly runs into limitations when we make direct comparison with OpenCL. Figure 5-22 demonstrates runtime comparison between OpenCL and TornadoVM for vector search with cosine similarity for vectors of size 2048 and varying number of vectors ranging from 1000 to 500,000. It is worth noting that TornadoVM on average performed 10 times slower than pure OpenCL implementation and ran into memory issues when the number of vectors approached 100,000.

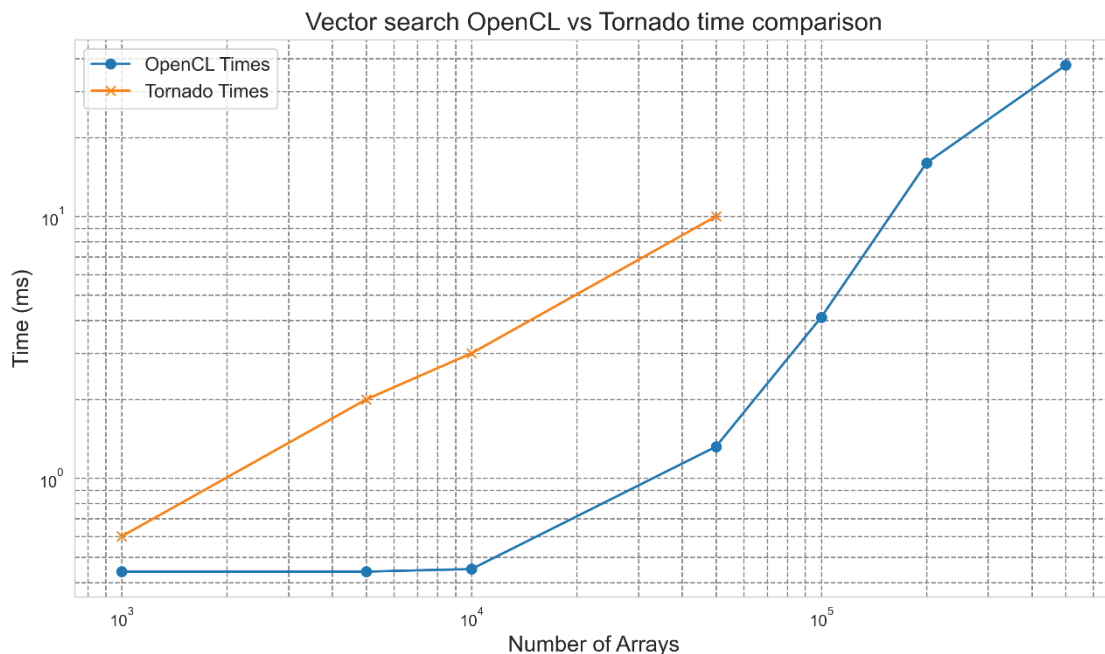


Figure 5-22. Vector search time comparison between Tornado and OpenCL

Despite TornadoVM shortcomings, the ideas presented clarify the approach in abstracting away heterogeneous hardware programming. Currently, the TornadoVM does not adapt to distributed systems, but it's one of the future works mentioned in the paper as an integration to multiple cloud-based VMs. There are other studies done in combining distributed computation with GPU or FPGA processing, e.g. SnurHAC [Jung2021] - a runtime for heterogeneous clusters with CUDA devices. It provides an illusion of a single GPU for multiple GPUs in a cluster by abstracting away the workload distribution and memory management. It achieved up to 28 times speed-up over a single GPU while easing the programming of the applications. Another example is EngineCL [Nozal20], which is a runtime system built on top of OpenCL to automatically distribute workload across multiple computing devices with excellent usability in mind. Although these tools simplify multiple device management, it is still expected from a developer to write a codebase using low-level C/C++ with CUDA or OpenCL.

Additionally, these tools are designed for use cases where hardware constraints are known, and the program is optimized during compilation time based on the specific application. We plan to extend either of these approaches to a distributed edge scenario with multiple interconnected devices with dynamic reconfiguration and load balancing while providing a high-level API for ease of application development.

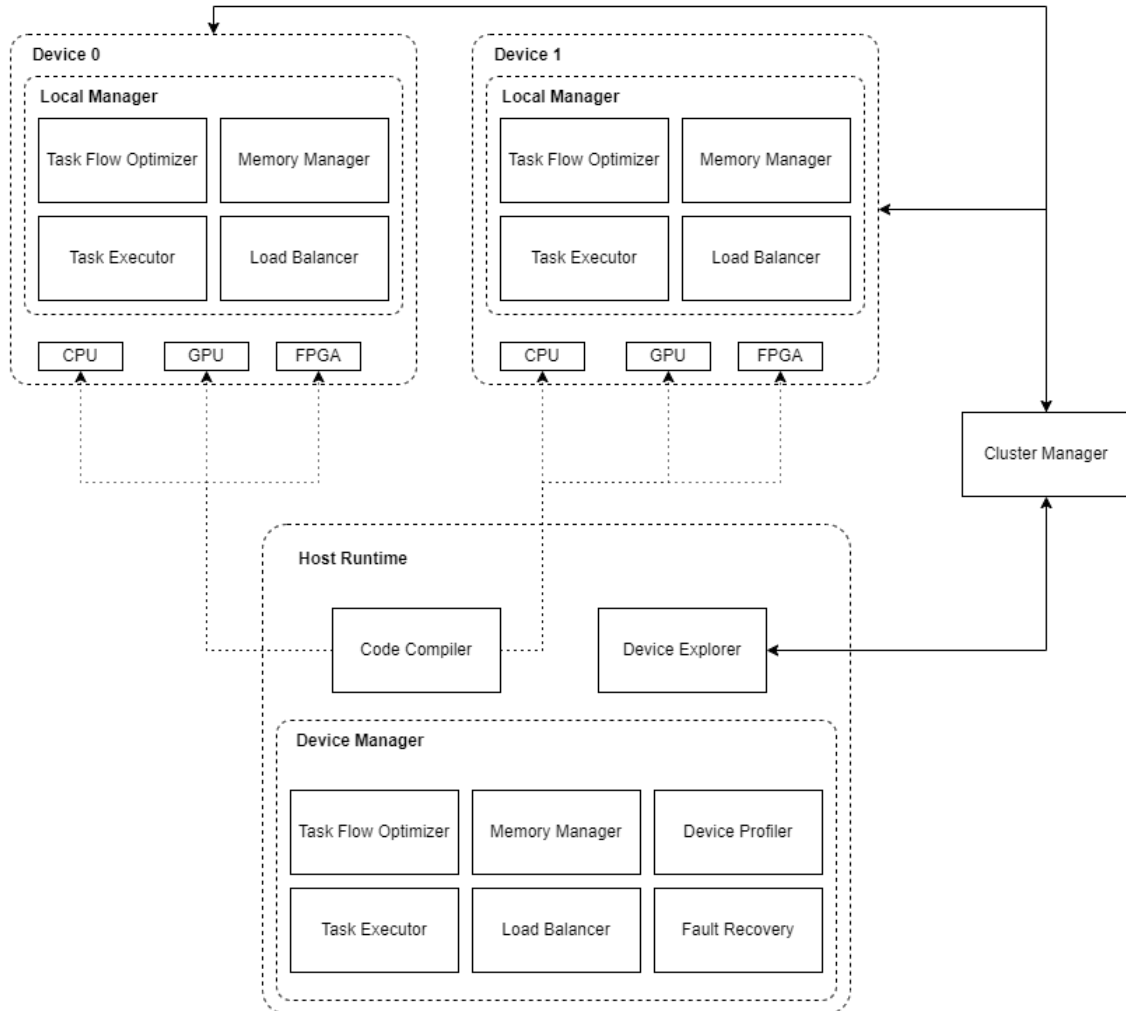


Figure 5-23. Distributed Execution on Abstracted Hardware

5.4.2.4.2 Design of Distributed Execution on Abstracted Hardware

Figure 5-23 illustrates our design for a distributed execution mechanism that can abstract the underlying hardware. Our design will contain several parts that will operate in a dynamic cluster of nodes. The main node, called Host Runtime, is responsible for orchestrating the workload among all available nodes and hardware on a high level. It comprises the following modules: Code Compiler, Device Explorer, and Device Manager:

- The Device Explorer module acts as a liaison between the Host Runtime and external Cluster Managers, such as Kubernetes, Apache Mesos, or Docker Swarm. It communicates with the Cluster Manager to discover available clusters, hardware devices, their configurations, and computational capabilities. This information is crucial for efficient workload distribution and resource allocation.
- The Code Compiler module is responsible for compiling application code into optimized kernels tailored for specific target devices within the heterogeneous

hardware environment. It ensures that the application is efficiently translated into executable code, taking advantage of the unique architecture and capabilities of each device.

- The Device Manager orchestrates the execution of the applications across heterogeneous hardware devices on a global level. It comprises several key modules:
 - The Task Flow Optimizer analyses the computational requirements of the application and dynamically optimizes the task distribution across available devices to maximize performance and minimize latency. It intelligently partitions tasks, considering factors such as device capabilities, data dependencies, and workload characteristics.
 - The Memory Manager module is responsible for efficient memory utilization across devices. It manages data movement, storage, and synchronization to minimize overhead and ensure optimal performance. It employs strategies like memory pooling, data compression, and memory sharing to mitigate memory constraints and enhance scalability.
 - The Device Profiler module gathers detailed performance metrics and profiling data from each device in real-time. It monitors factors like compute utilization, memory usage, power consumption, and network latency to provide insights into device performance and identify optimization opportunities. This information enables informed decision-making for task allocation and resource management depending on set KPIs to optimize for.
 - The Task Executor module is responsible for executing tasks on individual devices according to the optimized task flow determined by the Task Flow Optimizer. It manages task scheduling, parallel execution, and resource allocation to ensure efficient utilization of device resources and timely completion of tasks.
 - The Load Balancer module dynamically distributes workload among available devices to maintain balanced resource utilization and prevent bottlenecks. It continuously monitors device performance and workload conditions; reallocating tasks as needed to optimize throughput and minimize response times. It employs algorithms like round-robin, weighted load balancing, or predictive scheduling to achieve optimal workload distribution.
 - The Fault Recovery module is responsible for handling device failures and recovering from system disruptions to maintain application availability and reliability. It implements fault tolerance mechanisms such as checkpointing, task replication, and automatic failover to mitigate the impact of hardware failures or network disruptions. It ensures seamless operation in the face of unexpected events, preserving data integrity and minimizing downtime.

The device module, called Local Manager operates autonomously on each device, ensuring that tasks are optimized and executed efficiently at the local level without manual intervention or configuration by the developer. They collaborate to maximize the performance and resource utilization of individual devices within the heterogeneous hardware environment, contributing to the overall acceleration of the applications at the edge. They require minimum setup and package as standalone containers. The Local Manager consists of the following local level modules:

- The Task Flow Optimizer within the Local Manager is responsible for optimizing task execution on the local device level. It analyses the computational requirements of the

tasks assigned to the device and dynamically optimizes their execution to maximize performance and efficiency. This includes task partitioning, scheduling, and prioritization based on local device capabilities and resource availability.

- The Memory Manager module within the Local Manager optimizes memory usage and data handling on the local device. It manages memory allocation, data movement, and caching to minimize latency and maximize memory bandwidth utilization. It employs strategies like data locality optimization, memory pooling, and cache management to enhance performance and reduce memory overhead.
- The Task Executor within the Local Manager is responsible for executing tasks assigned to the local device. It manages task scheduling, parallel execution, and resource allocation within the constraints of the local hardware environment. It ensures efficient utilization of CPU, GPU, or FPGA resources, minimizing idle time and maximizing throughput for optimal task execution.
- The Load Balancer module within the Local Manager optimizes workload distribution and resource utilization on the local device. It monitors the computational load and resource availability on the device, dynamically adjusting task allocation to maintain balanced utilization and prevent resource contention. It employs load balancing algorithms to evenly distribute tasks among CPU cores, GPU threads, or FPGA resources, ensuring efficient utilization of hardware resources and optimal performance.

5.4.3 SmartEdge Processing Primitives

The SmartEdge processing primitives refer to fundamental operations that can be run on edge computing nodes that host SmartEdge Runtime. Such primitive operations provide the mechanism for Orchestrator in Section 4 to push the processing closer to edge devices. The include three types: graph stream query operators, tensor computations and sensor fusions.

5.4.3.1 Graph stream query operators

The graph stream query operators provide the core features for expressing data fusion operations in T5.1 and WP3 given that SmartEdge extend RDF data models for capturing sensor data interlinked with common sense and domain knowledge graphs.

- Basic graph matching: This operator is atomic operator in graph query language like SPARQL, called basic graph pattern. It provides the basic building block for filtering and graph pattern matching for more complicated query processing as well as some part of Recipe defined in D3.1
- Multiway stream join: A multiway stream join operator is responsible for correlating data from multiple streams based on certain join conditions. This is particularly useful in UCs where information from different sensor sources needs to be combined to form a unified view, such as combining radar data with output of object detections from camera in UC2.
- Aggregation: Aggregation operators summarize data streams, providing statistics like counts, averages, sums, and min/max values. This is essential for applications that monitor trends over time, such as counting queuing vehicles in UC2.
- Similarity search: Similarity search operators find similar elements in data streams, facilitating tasks such as object identifications. This operator uses metrics such as Euclidean distance or cosine similarity to quantify likeness between output embeddings of stream data elements, e.g detected objects in video frames.

5.4.3.2 Tensor computations

Tensor computations are operations performed on tensors, which are multi-dimensional arrays that generalize matrices to higher dimensions. They are fundamental to sensor fusion and DNN-based models to be deployed in edge devices

- **Matrix operations:** These operations include matrix addition, multiplication, and inversion, which are foundational for many machine learning algorithms. Efficient matrix computation at the edge can significantly speed up model training and inference processes.
- **Vector ranking/Similarity:** Vector ranking involves ordering vectors based on certain criteria, often used in processing components that deal with multimodal data stream such as T5.1, e.g scene understanding and object tracking. Similarity computations compare vectors to find how alike they are, which is crucial for tasks like clustering and classification.
- **Deep neural networks (DNNs):** DNN computations involve forward and backward passes through layers of neurons, enabling feature extraction and decision making. Edge devices with the capability to perform these computations can execute complex models such as object detection, segmentation and classification locally, thus supporting applications like image recognition and natural language processing.
- **Metric calculation for vectors:** Metric calculations for vectors involve computing the distance or similarity between vectors, which is essential for many machine learning tasks, such as k-nearest neighbors (k-NN) algorithms.

5.4.4 SmartEdge plugins

5.4.4.1 P4 Runtime plugin

The Smart-node Network Control Plane Layer (SNCPL) is a software module that runs within both the Access Point (AP) and the Swarm-node. In the context of an Access Point the SNCPL operates in coordination with the Service Layer and a global node registry in order to ensure a swift and seamless joining of required devices (actuators, sensors) to the swarm domain, based on prerequisites declared by the swarm application running at the Application Layer. The SNCPL

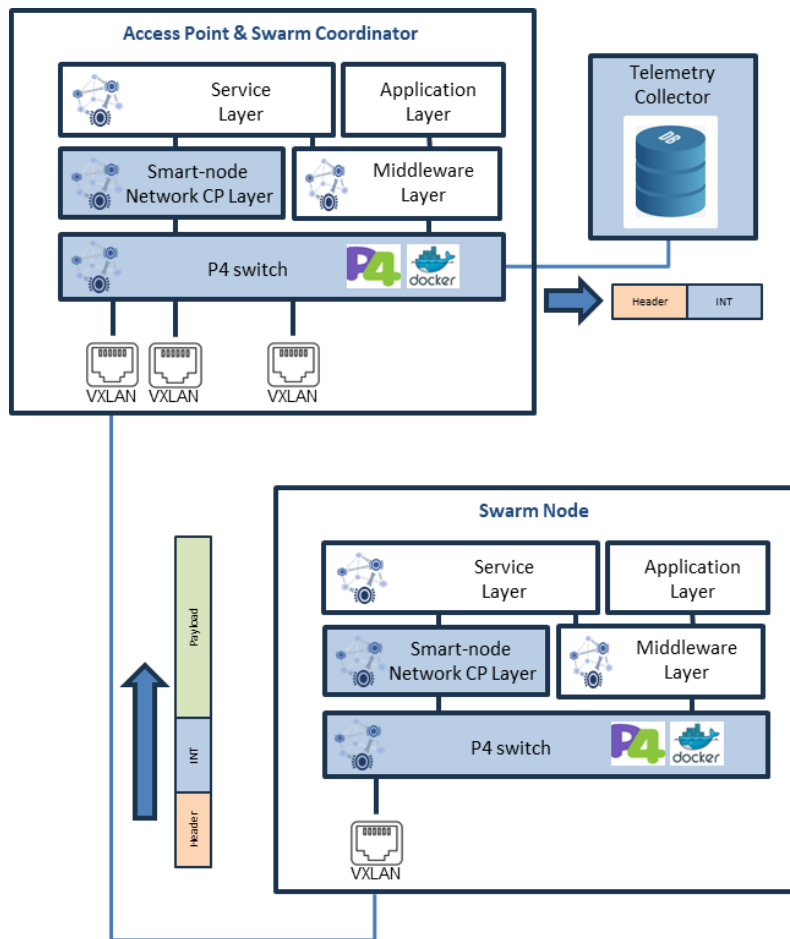


Figure 5-24. Architecture modules of a swarm application with one Smart-node and one Coordinator

uses socket communication over predefined Layer-Four port numbers for the purpose of handling the join and leave requests from the Smart-nodes in the swarm domain.

The SNCPL also interacts through Inter-Process Communication with processes of both higher and lower layers running within the same system to perform the verification of the Universal Unique IDentifier of the newly introduced device (swarm node), as well as the validation that the class of the device (that indicates the capabilities of the device) aligns with the application requirements.

Furthermore, The SNCPL handles the injection of Match-Action table entries in the programmable P4 switch running within the Access Point through an Apache Thrift Remote Procedure Call (RPC) interface. Installing the correct table entries enables the P4 programmable switch to properly route the data traffic among swarm nodes including unicast, multicast, and broadcast traffic. Moreover, the P4 switch running in the Access Point assumes the role of extracting any In-Band Telemetry metadata included within the data packets, before forwarding the packet to its destination. As this type of telemetry data is meant for the monitoring system that is overlooking the actual operational state of the swarm. The collected telemetry data depends on the class of the device that has generated the telemetry metadata and can include various vital parameters of the concerned device such as the loading of the CPU of the device, the remaining charge within the battery, the location of the device, and so on. The collected telemetry would then be transported to the Telemetry Collector which handles the processing and classification of the collected data to deduce visionary insights about the state of the current application and possibly provide recommendations for optimizing the effectiveness of current

working applications. By implementing a mechanism for reading the feedback from the telemetry system, developers of swarm applications can account for condition variabilities of the expected behavior, as well as a better interaction between hardware and software components involved in the current undergoing task.

While within the context of a Smart-node, the SNCPL communicates with its peer layer at the swarm layer using TCP communication for transmitting control messages regarding the membership of the swarm domain, so far three types of these messages have been tested in a limited scenario. Namely, the Join message through which a Smart-node expresses its will to join a swarm and take part in a specific application showing its capabilities. The second type of message is a Switch message, which indicates that a Smart-node is willing to handover its communication from one Access Point to another. The third type of message is the Leave message, which expresses a Smart-node will end its role in the current application and leaves the swarm.

All software components have been run and tested in a Linux environment running Ubuntu 20.04.

5.4.4.2 In-Network Machine Learning Attachment

WP4 uses automated frameworks to train, compile, map and deploy machine learning (ML) models within the network. These are used mainly for security purposes but can also be used to accelerate other types of network functions, particularly distributed ones.

The architecture of the distributed in-network computing framework is shown in Figure 5-25 below. The interface with WP5 is in the points marked (1) and (2) in the drawing. In (1), information about the implemented function and used devices is passed to the framework. In (2), information about the use-case network is being shared, i.e. the devices that are part of the swarm and their connectivity. This can be combined with information from the P4 runtime (5.4.4.4.1).

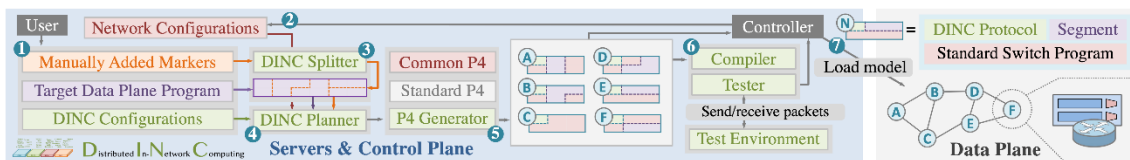


Figure 5-25. Architecture of Distributed In-Network Computing Framework

To support distributed in-network ML solutions, the mapping of a trained ML model is automatically generated and is merged in (1). A training dataset and features used need to be defined, as well as use case configurations. Note that existing P4 code, e.g., used for swarm management, needs to be available in order to be merged. Runtime updates of the model, e.g., due to data skew, are possible using P4Runtime. To support hitless updates, all the features need to be defined pre-deployment, as well as maximum table sizes.

The solution is aimed to support several types of P4 targets and has so far been tested on Linux software switch (BMv2, Dell IoT Gateway) and Intel Tofino. More information is available in [Zheng23].

5.4.4.3 Security Model of Swarm management

With the focus on building the security model of swarm management three use-cases have been analysed (UC-2, UC-3, UC-5) and security requirements have been refined into a model.

For the traffic management case, UC-2, the following threat model has been articulated:

1. No safety critical or real-time aspects. All tech deployed in SmartEdge is for optimisation purposes.
2. DoS issues are unclear at this time

The most interesting research question is reputation-based trust and pseudonymization (IDM-021):

- a. Due to GPRS, any supplier of location information should be allowed to assume a pseudonym, possibly multiple pseudonyms.
- b. Pseudonymous suppliers of data should be able to improve their reputation rating if their messages are confirmed by either post hoc observations or other suppliers. Also, if they lie, their reputation must sustain a hit.

Security requirements differ between the project demo and the prospective full product because the demo is much more tightly controlled by the project team than the product will ever be, so it was decided that the components not required for the project will be plugged up.

For the health-care use case, UC-5a, the security model is somewhat different.

1. Due to the safety-critical situation in health care, any multivendor environment has to include a provision for non-repudiation: medical data on which care decisions can be based should be signed by the equipment that produces them in a manner that does not require trust (SC-027).
2. Healthcare deals with masses of personal data, but secrecy is difficult to manage since domain experts routinely cross administrative boundaries. For example, patients can be referred to a consultant, who could be external to the assisted living support system and could even be outside the local network. However, the case is well served by detaching the data sets from the precise identity of the patient, i.e. by pseudonymisation (IDM-021, IDM-018). Confidentiality requirements still exist to protect communication between patient and edge server to prevent Advanced Persistent Threats (long-term monitoring of customers with the purpose to construct a statistical profile for commercial targeting).

We conclude that both safety-critical and non-safety critical cases require pseudonymity; however, in one case we also require a reputation management system, whereas in the other we only need to protect the user's identity. Digital signature services are not essential to one case, while being crucial to the other.

Another difference is the resource footprint. For smart sensors characteristic of medical applications, the computational and communication resources are much more limited due to the use of battery powered microcontrollers and a low-bandwidth, low-power communication infrastructure. By comparison the automotive case is not significantly restricted. However, the latency tolerance works in the opposite direction. In city traffic control, decisions should be made in the time frame of a moving vehicle, whereas patient care, even when dealing with emergencies is not on the single second scale, especially in the area of assisted living.

Speaking of the third use case, a smart factory, UC-3, we note that:

1. Like traffic control, real-time process operates in the closed loop, so **there is no real-time aspect** to the system

2. **Non-repudiation** (SC-027) is high on the agenda for this use case as well. The reason is similar to that for UC-5: the desire to be able to establish the root cause of failure based exclusively on fact (and not on excessive trust). The difference is that a smart factory robot is not constrained by power or bandwidth compared to the typical smart sensor for UC-5.
3. **No flexible identity management or pseudonymity** is required. In a smart factory all robots have a cryptographically protected digital ID and should use it in all cases.
4. Having said that, **reputation-based trust** is still desirable, since a factory can let an external actor operate on its premises and in this case the amount of trust should be established based on the track record. The difference from UC-2 is that actors are not pseudonymous, but completely identified.
5. ROS, the operating system for robots has very poor namespace management facilities, and so support for multiple security domains as regards confidentiality of data (IDM-021) is required. Dell would prefer to have a decentralised solution (perhaps not completely, but with a chain of trust rather than a single TTP).

In building the SMARTEDGE security infrastructure the following stages have been formulated:

1. Resource-limited digital signature scheme. We have developed a hash-based digital signature scheme which can be implemented on a microcontroller equipped with a hardware cryptographic accelerator, such as the one found in the Espressif ESP32. The scheme is asymmetric, requiring little power on the part of the smart sensor and a small transmit bandwidth, while the fog server executes more code and produces more data — without needing to trust each other. We intend to bring this new security protocol to bear on the swarm situation.
2. The same protocol can be adapted to reputation management. The difference is that a single account will serve a whole swarm with provisions made to ensure progress even when a counterparty stalls. This will require some new development, but it is justified by the fact that the same basic primitives will serve both use cases.
3. As a risk mitigation stage, if the zero-trust solution of the previous bullet point proves difficult within the constraints of a swarm, we will develop an approach based on a Swarm Trusted Third Party (STTP). This will be in line with Dell's approach to swarm management based on leader election. It would be logical to endow the leader with additional trust and manage the rest of the swarm that way. STTP would then be able to award reputation points to a pseudonym, trade points between pseudonyms, and maintain confidentiality of all transactions. The downside of the STTP solution is in its single point of failure, which requires additional protection and additional resources on the part of the elected leader, making the situation heterogeneous.

5.4.4.4 Mendix plugin

To make data generated by a SmartEdge swarm composable for Medix-based Swarm Intelligence apps in WP3, SmartEdge runtime provides a data exchange mechanism with Mendix, serving the data produced by SmartEdge runtime to Mendix. Via this, a Mendix process/recipe can federate data querying workload via SPARQL Federation service to a SmartEdge runtime.

On the other hand, to make data provided via OPC-UA in UC4 accessible for edge swarm nodes, e.g. robots running DDS and ROS2, the plugin also allows Mendix runtime to send the data to

SmartEdge runtime so that the data and devices behind that OPC-UA servers can be transparently part of the SmartEdge Swarm.

At first, the interaction between Mendix and SmartEdge runtime will be implemented with the focus on data elements related to UC3 and UC4 with the possibility to be extended for other OPC-UA companion specifications. Such modules might follow the declarative transformation approach of WP3 with some extensions of semantic DSL provided in Section 6.4.1. In particular, the shared data elements among Robots, PackML and Machine Vision companions, IEEE AuR ontology (see Section 3.3 of D3.1) and ROS 2 data structures will be aligned with this plugin.

5.4.4.5 Chunks & Rules

An open-source alternative to the Mendix runtime is the chunks and rules initiative, being developed by W3C's Cognitive AI community group, see the Chunks and Rules specification [Chunks and Rules].

The initiative builds upon decades of work in the Cognitive Sciences on cognitive architectures, and more specifically, John Anderson's work on [ACT-R](#) at CMU [ACT-R]. Knowledge graphs can be expressed as chunks using a lightweight notation. Each chunk is a collection of name/value pairs where values are names, numbers or sequences thereof. Behaviour is expressed using simple condition/action rules that operate on cognitive buffers holding single chunks and may invoke asynchronous operations for complex processing and delegating behaviour to actuators.

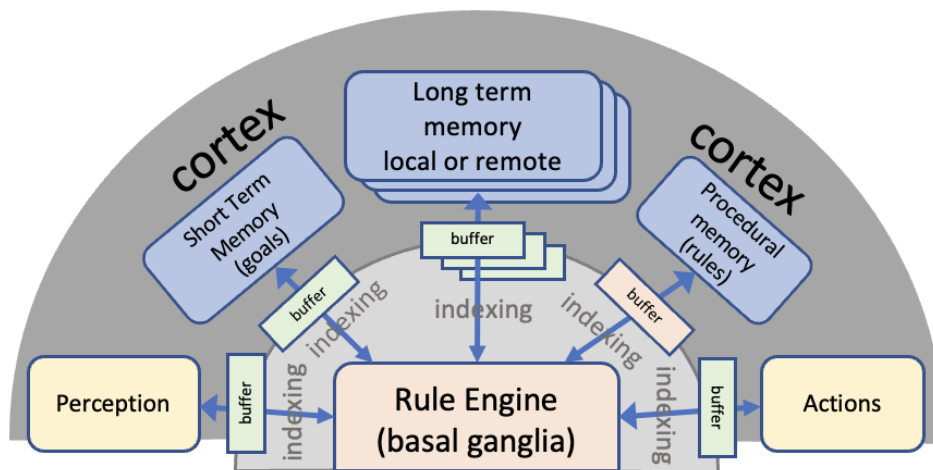
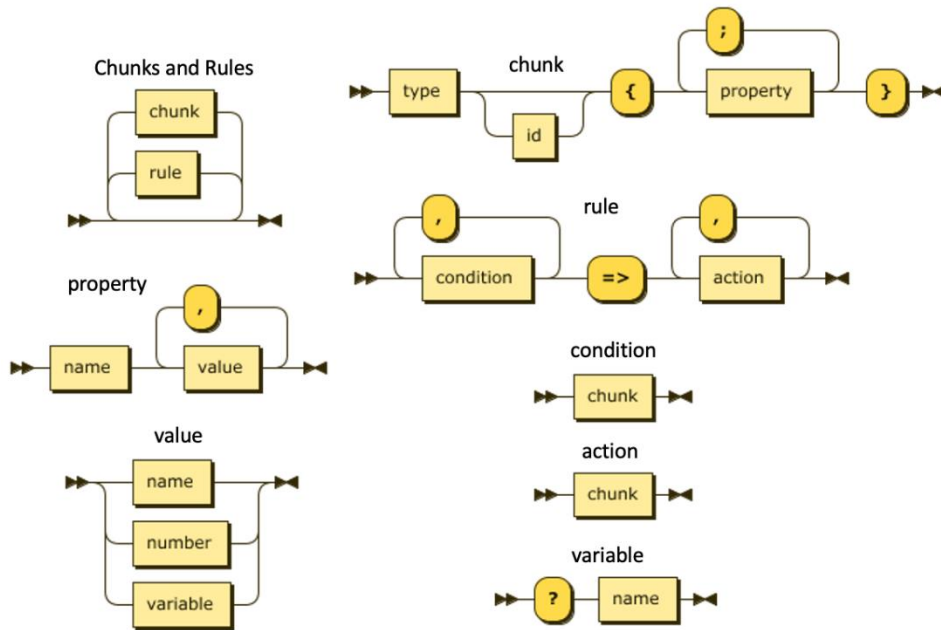


Figure 5-26. Cognitive Architecture for chunks & rules

Chunks & rules mimics characteristics of human cognition and memory, including spreading activation and the forgetting curve. There is a built-in set of operations along with the means for applications to define their own operations as needed. The syntax is easy to learn:



names beginning with “@” are reserved, e.g. @do for actions

Figure 5-27. Chunks & Rules Syntax as Railroad Diagrams

An open-source JavaScript implementation is available that can be run on suitable edge devices or in the cloud, e.g., as part of web pages or as part of NodeJS applications. Applications can define custom actions, e.g., to control conveyor belts, robot arms and other machinery. Rule execution runs independently and asynchronously from real-time control over actuators, see the [Robot demo using Chunks and Rules](#) [Robot Chunks Demo].

The framework lends itself to distributed models of control where you have multiple communicating agents that are executed on different processors, and threaded models of behaviour, where the completion of one thread triggers the next thread. Agent to agent communication is modelled in terms of perception and actuation, using application extensions. Agents are identified by names, leaving messaging to lower layers that know what protocols and standards are needed for each agent. In principle, this could be built upon W3C’s Thing Descriptions which use RDF to model the affordances and protocol bindings.

At the time of writing this section, we are still at an early stage in understanding the role of low-code for each of the use-cases, with the exception of use case 4, for which we have examples of how recipes could apply.

Use case 3 describes the use of mobile robots for transferring work pieces between conveyor belts and racks that move the work pieces from one manufacturing cell to another, see D2.2 Section 2.1.19.3. In principle, we could use chunks and rules to specify the behaviour of the mobile robots in terms of event triggered tasks, where each task is modelled using a set of rules that operate on the chunks defining the current state.

The rules control robot actuators via invoking ROS (Robot Operating System) controllers using application defined extensions for @do actions in the chunks and rules language. Here is an illustrative example for directing a robot arm to move to a given position, orientation and gripper separation, and then grasp the workpiece, when the workpiece on conveyor belt named *belt1* reaches the end of the belt and stops:


```

# move robot arm when belt1 is full
full {thing belt1} =>
    robot {@do move; x -120; y -75; angle -180; gap 40; step 1}

# move robot arm into position to grasp work piece
after {step 1} =>
    robot {@do move; x -170; y -75; angle -180; gap 30; step 2}

# move work piece to near the mobile rack
after {step 2} =>
    goal {@do clear},
    robot {@do grasp},
    robot {@do move; x -80; y -240; angle -90; gap 30; step 3}

```

The “step” argument used to trigger the behaviour to be executed when the operation has completed. For this, the application code waits for ROS to signal that the operation has finished, and then queues a chunk to the cognitive buffer to trigger a matching rule. Variables can be used in place of explicit values. The variables are scoped to the rule and bound when matching the rule’s conditions.

Perception is the process for interpreting sensory information and building live models of the current state of the world as collections of chunks (*aka* knowledge graphs). Chunk rules can also operate on these models using built-in CRUD based operations (create, read, update, delete) as well as through application defined extensions for more complex operations that use a lower-level chunk API. Events can be modelled as single chunks that trigger rules when injected into chunk buffers. Chunks & Rules supports prioritised queues for chunk buffers as the basis for handling urgent events.

Chunks can be viewed as a higher-level representation than RDF triples. Each chunk corresponds to a set of triples with the same subject vertex in the triple graph. Further details can be found at the [W3C Cognitive AI Community Group](#).

5.4.5 Application-support Adaptors/Connectors

5.4.5.1 Metric Reporting & Visualization Tools

SmartEdge provides an integrated Metric Reporting and Visualization tool, offering a streamlined approach for reporting and gathering metrics from various components within the SmartEdge ecosystem. The aim of the component is to monitor the SmartEdge system status and its configuration in order to detect possible misbehaviour or misconfigurations at an early stage. The same can be used for SmartEdge applications. In addition, by using the Metrics component and based on the data collected, SmartEdge and applications can be optimised. Illustrated in Figure 4-28 is the Architecture of Metric Reporting and Visualization, comprising the following key modules:

- **Metrics Client:** The Metrics Client is a library tailored to the technology stack of the corresponding SmartEdge Component. It actively monitors and retrieves the required metrics, transmitting them via HTTP REST API to the Metrics Reporting Server. These metrics consist of data-time series, captured and reported periodically within a configurable interval. Beyond the timestamp, additional contextual data such as device identifier and location can accompany the metric type and value. In instances where

connectivity between the SmartEdge device and the Metrics Server is temporarily unavailable, the SmartEdge component can cache the metrics data for a defined period. Subsequently, it reports the cached metrics to the server once connectivity is reestablished. The capacity for data caching should be adjustable. The client has the capability to send the cached metrics in a bulk request (or multiple requests) to the metrics server. Moreover, if a SmartEdge device within a swarm faces challenges in establishing an HTTP connection to the metrics server, potentially due to limited processing capabilities, another device within the swarm, such as the swarm coordinator, can function as a Metrics client. It can then relay the metrics from devices in the swarm to the metrics server.

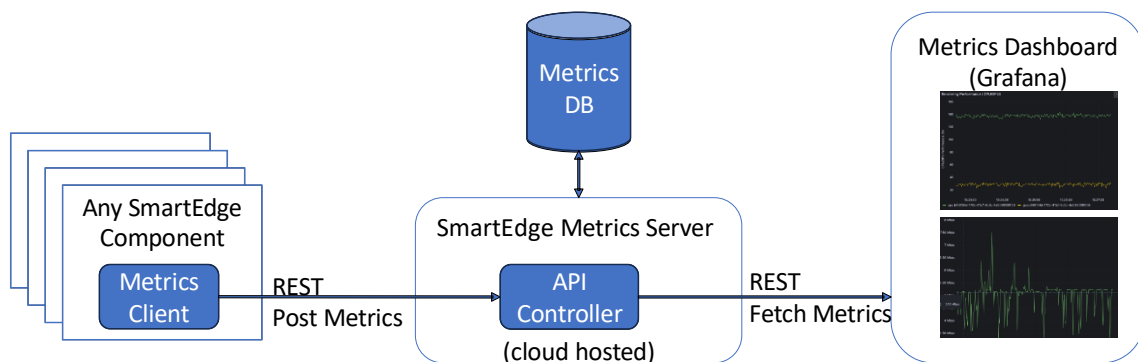


Figure 5-28. Metric Reporting and Visualization

- Metrics Server:** The Metrics Server serves as a centralized component responsible for collecting all metrics sent by SmartEdge devices through the REST API. It receives requests via HTTP POST, with the metrics embedded in the HTTP body encoded in JSON format. Subsequently, the Metrics Server stores this data as time-series in the database. Additionally, it provides a secondary API that allows retrieval of the stored data. This functionality proves valuable for tasks such as visualization in the dashboard or conducting further analysis on the collected metrics.
- Metrics Dashboard:** The Metrics Dashboard establishes a connection with the Metrics Server to retrieve collected data, presenting them in an array of comprehensive widgets, as illustrated in Figure 5-29. This dashboard incorporates a dynamic feature allowing users to seamlessly add new widgets for visualizing specific metrics or a combination of multiple metrics. Moreover, the dashboard provides extensive options for filtering historical collected metrics, enhancing the flexibility and depth of analysis.



Figure 5-29. Metrics Dashboard

- **Example Metric Types:** While each use case may define a unique set of metrics, there are metrics relevant across all use cases. Examples include CPU Usage, CPU Memory, GPU Usage, GPU Memory, Latency (various types), Uplink Throughput, Downlink Throughput, Packet Loss, and Jitter.
 - **Following some example Metrics of a selected use case to illustrate where metrics can help to monitor and improve the system behavior (UC2 - Smart Traffic Control):**
 - **Node to node messaging latency:** The communication method, namely DDS, MQTT, or C-ITS V2X, could also affect latency. An example is the latency of data sent from Sensor Nodes to Controller Nodes or vice-versa. UC2 nodes also include the *original timestamp* (message generation time) along with the published JSON data, thus making it easy for the target node's code to calculate network latency. A tentative deadline of *100 to 200 milliseconds* is defined for UC2 node to node messages. As an example, the **SmartEdge Metric Client** could show the percentage ratio of messages missing the latency deadline.
 - **Task execution time (message processing time):** This metric refers to the computational load on nodes' *edge devices*. A tentative *deadline* of **1 to 5 milliseconds** is defined depending on each task of UC2. The **SmartEdge Metric Client** could for example measure the ratio of tasks that miss their deadline. Example tasks to observe include:
 - **Traffic control decision making for the option zone:** The time taken by a Controller Node to process a combination of related incoming sensing messages (from vehicles of a certain road-segment) before making a traffic control decision for the option zone.

- **Queue length per lane calculation:** The time taken by the indicator node to compute queue length of vehicles in each lane of a road-segment.
- **Message processing drop rate (percentage of messages):** Because sensor messages in UC2 typically arrive at around 10 Hz rate, the target node edge device (e.g. Controller) may have to drop some messages if it does not manage to process all messages from all surrounding sensor nodes in good time.
- **Camera's object detection rate (how many updates per second):** Each detection includes object's classification together with its new location and speed. Object detection updates should be at least 10 times per seconds to enable safety applications such as the option zone control.

5.4.5.2 Remote Rendering and Streaming Pipeline (FhG?, ready for reviewing)

The Remote Rendering and Streaming pipeline by Fraunhofer FOKUS depicted in Figure 5-30 offers a powerful solution for rendering and streaming compute-intensive 3D applications and Metaverse experiences on nearly any device. Even devices with limited graphical processing capabilities, can access high-quality Metaverse experiences with ease. This is made possible by offloading the compute-intensive 3D graphics processing to remote servers that run on-premise, on the edge of a 5G network, or in the cloud. The rendered views are then streamed to end devices for display.

In remote rendering, the view port of the virtual camera within the 3D experience is rendered headlessly on the server and encoded as a video/audio stream, which is then delivered to the user device via WebRTC, which only needs to play the stream. User interactions, which can vary between different device classes, such as mouse/keyboard input on desktops, touch inputs on mobile devices, remote control on TVs, and motion control on AR/VR displays, must be captured and sent to the remote rendering server, which triggers the received events on the underlying rendering engine as if they were received from a connected input device.

In SmartEdge, the Remote Rendering and Streaming will be used to run extensive simulations and tests in photorealistic virtual environments supporting many of the SmartEdge use cases especially use cases 1 and 2.

In addition, the Remote Rendering and Streaming pipeline integrates with the Metrics Reporting and Visualization Tool introduced in the previous section. Performance metrics such as CPU and GPU performance, memory usage, latency, and bitrates, among others, are reported and visualized in the Metrics Dashboard.

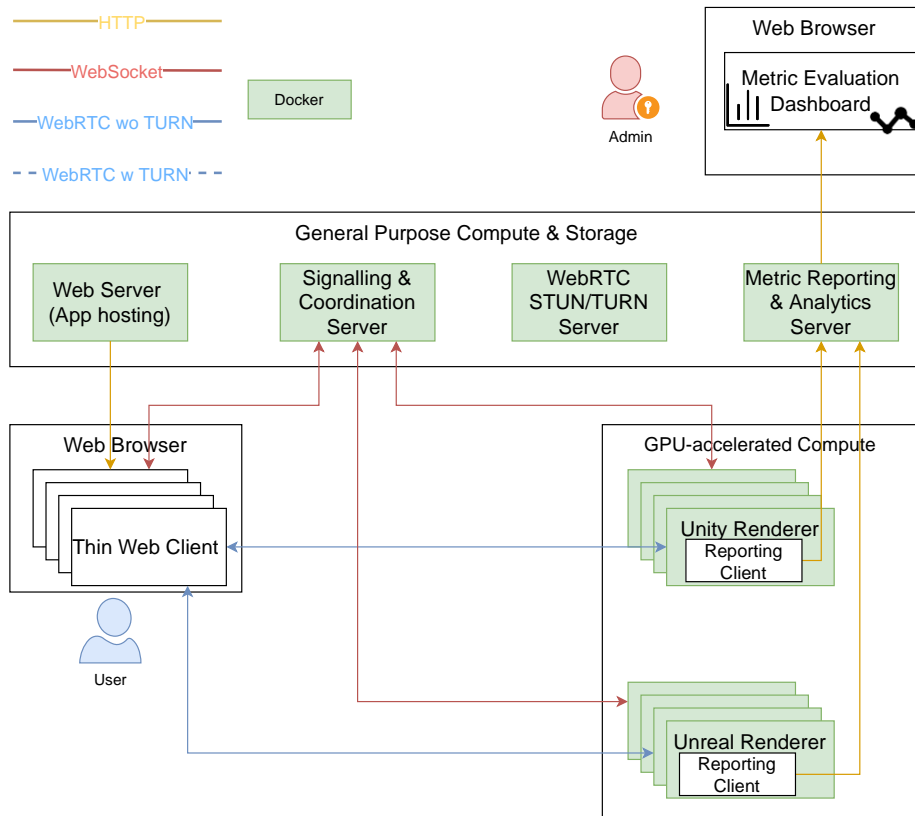


Figure 5-30. Remote Rendering and Streaming Architecture

5.4.5.3 Ego-vehicle centric Visualization (TUB)

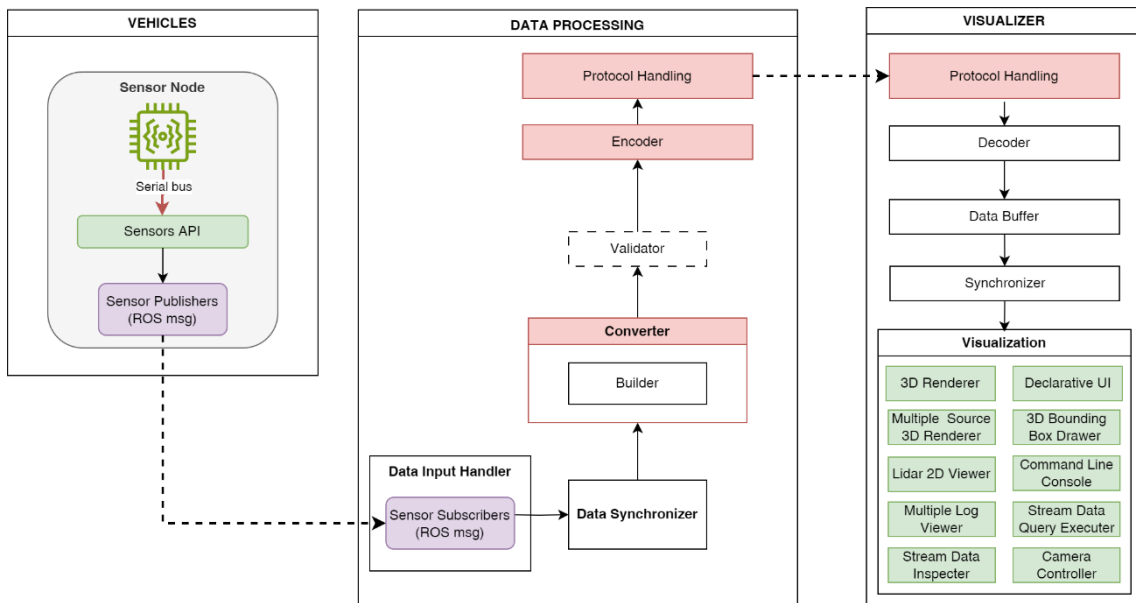


Figure 5-31. Architecture of Multi-view Visualizer tool

The Multi-view Visualizer is a tool that supports visualizing and interacting with live data from vehicles in a synchronized manner. The tool consists of two main components: Data Processing and Visualizer as shown in Figure 5-31.

Data Processing: The Data Processing component is a customized iteration of the XVIZ Server, originally developed by Uber. It functions as a backend infrastructure element responsible for storing, processing, and serving data. The data from SmartEdge agents could stream to the Data Processing through the publish-subscribe mechanism of ROS and be added to the Synchronizer to synchronize data from multiple sources. Moreover, this component offers an interface through which data can be accessed and retrieved for visualization purposes. By enabling efficient data streaming and supporting real-time updates, the server enables clients, such as web-based visualizers or other applications, to receive and process data in a synchronized manner. This capability facilitates real-time monitoring and analysis of autonomous vehicle (AV) sensor data, streamlining tasks like debugging, testing, and simulation of autonomous systems.

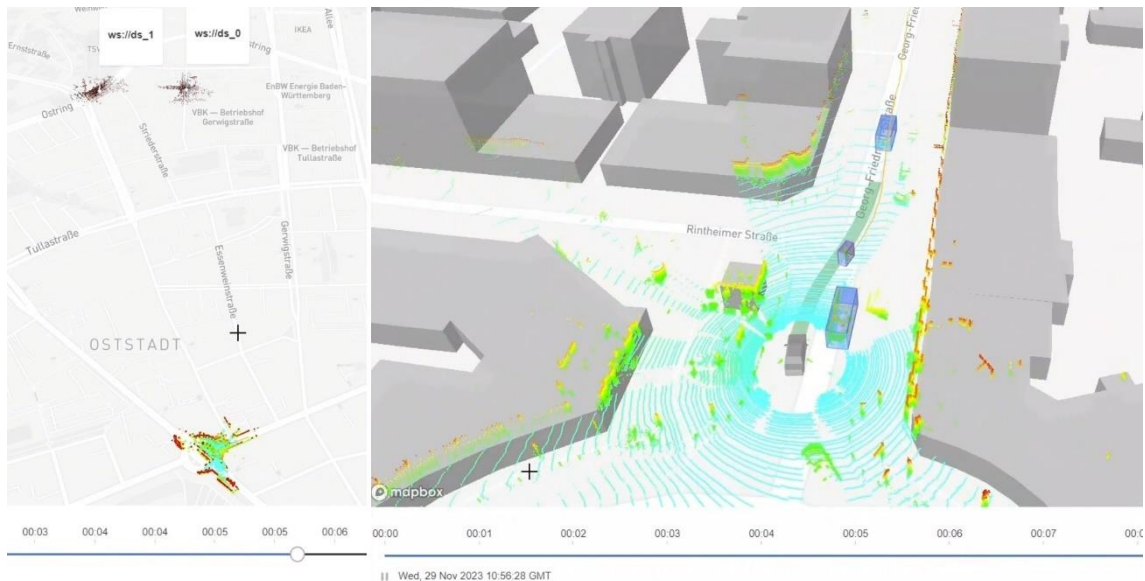


Figure 5-32. Multi-view Visualizer Screenshot

Visualizer: The Visualizer component is a web-based tool built upon an open-source framework developed by Uber's Visualization team, known as Streetscape.gl. Leveraging the robust capabilities of Streetscape.gl, the Visualizer inherits powerful features tailored for visualizing and exploring geospatial data, particularly within urban environments and in the context of autonomous vehicles (AVs). This component consumes data transmitted by the *Data Processing* component to visualize and analyse information captured by AV sensors, including lidar, cameras, and radar. By overlaying sensor data onto a 3D map, the Visualizer offers a comprehensive view of the surrounding environment, enhancing the comprehension of complex urban scenes. In the context of this project, the Visualizer component plays a crucial role in visualizing autonomous data streamed from various sources simultaneously, thus facilitating a visual assessment of data exchange capabilities among vehicles. A screenshot depicted in Figure 5-32 showcases the visualizer.

5.4.5.4 Swarm Visualization

One approach to visualizing swarms is to use web pages, exploiting HTML5's CANVAS element for 2D, 2.5D or potentially even 3D renderings. 2.5D isometric rendering is a good choice offering fast rendering speeds and game like graphics. There are plenty of examples of isometric images:

- <https://www.bing.com/images/search?q=isometric+city+buildings>

- <https://www.bing.com/images/search?q=isometric+factory+floor>

The starting point is a set of 3D models that can be imaged from different directions to create the image tiles used at run-time. The complete scene is composed from the components, e.g. for a city scene, this involves the road, road junctions, traffic lights, cars, trucks, pavements, buildings, street lights, trees, pedestrians, etc.

One tool chain for this is the Blender 3D editor plus a Python script to generate the image tiles used at run-time. This can then be used to support panning, zooming and rotation (limited to say 45-degree increments). Whilst ERCIM can help with the graphics resources, we would need your input at least as a starting point.

If the swarm state is held on an edge server or cloud server, the state can be streamed in real-time to the web page via WebSockets. The same socket connection can be also used to send back messages as a means for users to interact with the swarm via affordances in the web page. An example could be tapping on the visual representation of a vehicle to ask for information on its current speed.

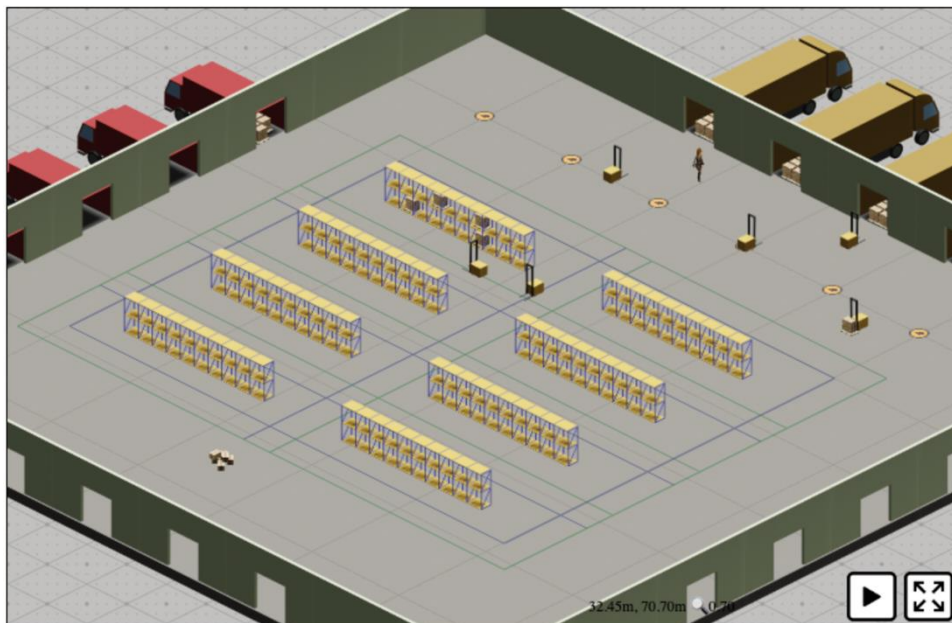


Figure 5-33. SimSwarm – screenshot of smart warehouse demo with robot forklifts, see: <https://www.w3.org/Data/demos/chunks/warehouse/>

The rendering process involves an HTML5 call back with a high precision time stamp. This can be driven by a message stream that specifies component position, scale, orientation, velocity and potentially acceleration, enabling the renderer to compute the state for the given time stamp. The scene components are rendered from back to front using a spatial sort that is updated as the scene changes.

Special care is needed to properly render situations where one object is partially inside another. One example is a forklift moving its forks into or out of a pallet. In my smart warehouse demo, I split the 3D model of the pallet into top and bottom pieces so that the bottom is rendered before the forks, followed by rendering the top of the pallet. I used off-screen compositing for rendering

the forklift and its payload when moving into or out of a docking bay. Similar approaches would be used for rendering a car driving under a bridge.

6 CONCLUSIONS

This deliverable D5.1 shows the first step towards achieving **Objective 5** of the SmartEdge project, namely, building *Low-code Programming Tools for Edge Intelligence* providing (i) semantic driven multimodal stream fusion for Edge devices; (Section 2) (ii) swarm elasticity via Edge-Cloud Interplay (Section 3) (iii) adaptive coordination and optimization; (iv) cross-layer toolchain for Device-Edge-Cloud Continuum. Each section presents different technologies and components, but they share the declarative programming and data models as the integration points for the overall design of the overall toolchain.

D5.1 reports the design outcome together with preliminary studies on how to build consistent components of the set of low-code programming. The design not only takes five KPIs as the guiding criteria but analyses several baselines of the current state of practice (Section 1.2). Moreover, along with the design realization, some initial studies and implementations were carried with some positive outcomes towards some subsets of UCs. This sets first steps towards the implementation of the first version of SmartEdge tool chain for the first milestone as well as the second deliverable of WP 5, namely D5.2. Aligning with the starting of WP6, D5.1 provides the design reference for integrating WP5 with other WPs in realization and validation of five targeted UCs.

REFERENCES

- [Aberer01] K. Aberer, "P-Grid: A self-organizing access structure for P2P information systems", in Cooperative Information Systems: 9th International Conference, CoopIS 2001 Trento, Italy, September 5–7, 2001 Proceedings, vol. 9, Springer Berlin Heidelberg, pp. 179-194.
- [ACT-R] "ACT-R Research Group, CMU", last modified 2023, URL: <http://act-r.psy.cmu.edu/>
- [Anh18] A. Le-Tuan, C. Hayes, M. Wylot, and D. Le-Phuoc. Rdf4led: An rdf engine for lightweight edge devices. In IOT '18, 2018.
- [Anh19] A. Le-Tuan, D. Hingu, M. Hauswirth, and D. Le-Phuoc. Incorporating blockchain into rdf store at the lightweight edge devices. In Semantic '19, 2019
- [Anh21] Le Tuan, Anh, et al. "VisionKG: Towards A Unified Vision Knowledge Graph." ISWC (Posters/Demos/Industry). 2021.
- [Auer17] Auer, Sören, et al. "Dbpedia: A nucleus for a web of open data." international semantic web conference. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007.
- [Balazinska04] M. Balazinska, H. Balakrishnan, and M. Stonebraker. Contract-based load management in federated distributed systems. In NSDI'04, 2004
- [Biffi20] Biffi, Leonardo Josué, et al. "ATSS deep learning-based approach to detect apple fruits." Remote Sensing 13.1 (2020): 54.
- [Bowden22] Bowden, David, and Diarmuid Grimes. "Intelligent Image Compression Using Traffic Scene Analysis." Irish Conference on Artificial Intelligence and Cognitive Science. Cham: Springer Nature Switzerland, 2022.
- [Carion20] Carion, Nicolas, et al. "End-to-end object detection with transformers." European conference on computer vision. Cham: Springer International Publishing, 2020.
- [Chen21] Chen, Qiang, et al. "You only look one-level feature." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2021.
- [Chen23] Chen, Qiang, et al. "Group detr: Fast detr training with group-wise one-to-many assignment." Proceedings of the IEEE/CVF International Conference on Computer Vision. 2023.
- [Chen19] Chen, Tianshui, et al. "Knowledge-embedded routing network for scene graph generation." Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition. 2019.
- [Chunks and Rules] "Chunks and Rules Specification", W3C Cognitive AI Community Group, last modified 04 January 2024, URL: <https://w3c.github.io/cogai/chunks-and-rules.html>
- [Cong23] Cong, Yuren, Michael Ying Yang, and Bodo Rosenhahn. "Reltr: Relation transformer for scene graph generation." IEEE Transactions on Pattern Analysis and Machine Intelligence (2023).
- [Cudre-Mauroux13] Philippe Cudré-Mauroux, Iliya Enchev, Sever Fundatureanu, Paul Groth, Albert Haque, Andreas Harth, Felix Leif Keppmann, Daniel P. Miranker, Juan F. Sequeda, Marcin Wylot: NoSQL Databases for RDF: An Empirical Evaluation. ISWC (2) 2013: 310-325.
- [Cui22] Cui, Yu, and Moshir Farazi. "VReBERT: a simple and flexible transformer for visual relationship detection." 2022 26th International Conference on Pattern Recognition (ICPR). IEEE, 2022.

- [Dai16] Dai, Jifeng, et al. "R-fcn: Object detection via region-based fully convolutional networks." *Advances in neural information processing systems* 29 (2016).
- [Danh11] D. Le-Phuoc, M. Dao-Tran, J. X. Parreira, and M. Hauswirth. A native and adaptive approach for unified processing of linked streams and linked data. In *ISWC'11*, pages 370–388, 2011.
- [Danh15] D. Le-Phuoc, M. Dao-Tran, C. Le Van, A. Le Tuan, T. T. N. Manh Nguyen Duc, and M. Hauswirth. Platform-agnostic execution framework towards rdf stream processing. In *RDF Stream Processing Workshop at ESWC2015*, 2015.
- [Danh17] D. Le-Phuoc. Operator-aware approach for boosting performance in RDF stream processing. *J. Web Sem.*, 42:38–54, 2017.
- [Danh18] D. Le-Phuoc. Adaptive optimisation for continuous multi-way joins over rdf streams. In *Companion Proceedings of the The Web Conference 2018, WWW '18*, pages 1857–1865, 2018.
- [Danh21] Le-Phuoc, Danh, Thomas Eiter, and Anh Le-Tuan. "A scalable reasoning and learning approach for neural-symbolic stream fusion." *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. No. 6. 2021.
- [Dell17] D. Dell'Aglio, D. L. Phuoc, A. Le-Tuan, M. I. Ali, and J.-P. Calbimonte. On a web of data streams. In *DeSemWeb@ISWC*, 2017.
- [Denny14] Vrandečić, Denny, and Markus Krötzsch. "Wikidata: a free collaborative knowledgebase." *Communications of the ACM* 57.10 (2014): 78-85.
- [Dias2019] Vinicius Dias, Carlos HC Teixeira, Dorgival Guedes, Wagner Meira, and Srinivasan Parthasarathy. 2019. Fractal: A general-purpose graph pattern mining system. In *Proceedings of the 2019 International Conference on Management of Data*. 1357–1374.
- [Dominik23] Kreuzberger, Dominik, Niklas Kühl, and Sebastian Hirschl. "Machine learning operations (mlops): Overview, definition, and architecture." *IEEE Access* (2023).
- [Duc21] Manh Nguyen Duc, Anh Lê Tuấn, Manfred Hauswirth, Danh Le Phuoc: Towards autonomous semantic stream fusion for distributed video streams. *DEBS 2021*: 172-175.
- [Fumero19] Juan Fumero, et al. "Dynamic application reconfiguration on heterogeneous hardware". *VEE 2019: Proceedings of the 15th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution* (2019): 165–178
- [Girshick15] Girshick, Ross. "Fast r-cnn." *Proceedings of the IEEE international conference on computer vision*. 2015.
- [Grubenmann18] T. Grubenmann, A. Bernstein, D. Moor, and S. Seuken. Financing the web of data with delayed-answer auctions. In *WWW '18*, 2018
- [Haller19] A. Haller, K. Janowicz, S. J. D. Cox, M. Lefrançois, K. Taylor, D. Le-Phuoc, J. Lieberman, R. García-Castro, R. Atkinson, and C. Stadler. The modular SSN ontology: A joint W3C and OGC standard specifying the semantics of sensors, observations, sampling, and actuation. *Semantic Web*, 10(1):9–32, 2019.

- [Hong21] Hong, Jinyung, and Theodore P. Pavlic. "Representing Prior Knowledge Using Randomly, Weighted Feature Networks for Visual Relationship Detection." arXiv preprint arXiv:2111.10686 (2021).
- [Hornung13] A. Hornung, K.M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees" in *Autonomous Robots*, 2013; DOI: 10.1007/s10514-012-9321-0.
- [Hussein23] Rana Hussein, Alberto Lerner, André Ryser, Lucas David Bürgi, Albert Blarer, Philippe Cudré-Mauroux: GraphINC: Graph Pattern Mining at Network Speed. *Proc. ACM Manag. Data* 1(2): 184:1-184:28 (2023).
- [ISD24] [dataset] <https://www.ncdc.noaa.gov/isd>
- [Jia23] Jia, Ding, et al. "Detrs with hybrid matching." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023.
- [Jiang23] Jiang, Bowen, and Camillo Taylor. "Hierarchical Relationships: A New Perspective to Enhance Scene Graph Generation." *NeurIPS 2023 Workshop: New Frontiers in Graph Learning*. 2023.
- [Jicheng23] Yuan, Jicheng, et al. "VisionKG: Unleashing the Power of Visual Datasets via Knowledge Graph." arXiv preprint arXiv:2309.13610 (2023).
- [Jung21] Jaehoon Jung, et al. "SnuRHAC:ARuntimeforHeterogeneousAcceleratorClusterswithCUDAUnifiedMemory". *HPDC '21: Proceedings of the 30th International Symposium on High-Performance Parallel and Distributed Computing* (2021): Pages 107–120
- [Kien21] Kien-Tran, Trung, et al. "Fantastic Data and How to Query Them." *NeurIPS (Workshop on Data-Centric AI)*, 2021.
- [Kirillov23] Kirillov, A., Mintun, E., Ravi, N., Mao, H., Rolland, C., Gustafson, L., ... Girshick, R. (2023). *Segment Anything*. arXiv:2304.02643.
- [Kundu23] Kundu, Sanjoy, and Sathyanarayanan N. Aakur. "IS-GGT: Iterative Scene Graph Generation With Generative Transformers." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023.
- [Kurt08] Bollacker, Kurt, et al. "Freebase: a collaboratively created graph database for structuring human knowledge." *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*. 2008.
- [Lee24] Sangjin Lee, Alberto Lerner, Philippe Bonnet, Philippe Cudré-Mauroux: Database Kernels: Seamless Integration of Database Systems and Fast Storage via CXL. *CIDR 2024*.
- [Lin17] Lin, Tsung-Yi, et al. "Focal loss for dense object detection." *Proceedings of the IEEE international conference on computer vision*. 2017.
- [Liu22] Liu, Shilong, et al. "Dab-detr: Dynamic anchor boxes are better queries for detr." arXiv preprint arXiv:2201.12329 (2022).
- [Liu23] Liu, Shilong, et al. "Grounding dino: Marrying dino with grounded pre-training for open-set object detection." arXiv preprint arXiv:2303.05499 (2023).

- [Manh19] Nguyen-Duc, M., Le-Tuan, A., Calbimonte, J.P., Hauswirth, M., Le-Phuoc, D.: Autonomous rdf stream processing for iot edge devices. In: JIST 2019, pp. 304–319. Springer, Cham (2019)
- [Manh22] Nguyen-Duc, Manh, et al. "SemRob: Towards Semantic Stream Reasoning for Robotic Operating Systems." arXiv preprint arXiv:2201.11625 (2022).
- [Mar23] Hirzel, Martin. Low-Code Programming Models. Commun. ACM 66(10): 76-85 (2023)
- [Munshi09] A. Munshi, The OpenCL specification, 1, IEEE, Stanford, CA, USA, 2009.
- [Nozal20] Raul Nozal, et al. "EngineCL: Usability and Performance in Heterogeneous Computing". Future Generation Computer Systems 107 (2020): Pages 522-537
- [Naphade19] Naphade, Milind, Zheng Tang, Ming-Ching Chang, David C. Anastasiu, Anuj Sharma, Rama Chellappa, Shuo Wang et al. "The 2019 AI City Challenge." In CVPR workshops, vol. 8, p. 2. 2019.
- [Onos24] <https://opennetworking.org/onos/>
- [Pang19] Pang, Jiangmiao, et al. "Libra r-cnn: Towards balanced learning for object detection." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019.
- [Ranftl20] Ranftl, René, et al. "Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer." IEEE transactions on pattern analysis and machine intelligence 44.3 (2020): 1623-1637.
- [Redmon15] Redmon, J., Divvala, S. K., Girshick, R. B., & Farhadi, A. (2015). You Only Look Once: Unified, Real-Time Object Detection. CoRR, abs/1506.02640. Retrieved from <http://arxiv.org/abs/1506.02640>
- [Ren15] Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." Advances in neural information processing systems 28 (2015).
- [Ren24] Ren, T., Liu, S., Zeng, A., Lin, J., Li, K., Cao, H., ... Zhang, L. (2024). Grounded SAM: Assembling Open-World Models for Diverse Visual Tasks. arXiv [Cs.CV]. Retrieved from <http://arxiv.org/abs/2401.14159>
- [Rezatofighi19] Rezatofighi, Hamid, et al. "Generalized intersection over union: A metric and a loss for bounding box regression." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019.
- [Robot Chunks Demo] "Robot demo using Chunks & Rules", last modified 02 Jul 2020, W3C Cognitive AI Community Group, URL: <https://www.w3.org/Data/demos/chunks/robot/>
- [Ryser22] André Ryser, Alberto Lerner, Alex Forencich, Philippe Cudré-Mauroux: D-RDMA: Bringing Zero-Copy RDMA to Database Systems. CIDR 2022.
- [Schneider22] Patrik Schneider, Daniel Alvarez-Coello, Anh Le-Tuan, Manh Nguyen Duc, Danh Le Phuoc: Stream Reasoning Playground. ESWC 2022: 406-424.
- [Shaoqing15] Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." Advances in neural information processing systems 28 (2015).

- [Speer17] Speer, Robyn, Joshua Chin, and Catherine Havasi. "Conceptnet 5.5: An open multilingual graph of general knowledge." Proceedings of the AAAI conference on artificial intelligence. Vol. 31. No. 1. 2017.
- [Tang19] Tang, Kaihua, et al. "Learning to compose dynamic tree structures for visual contexts." Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. 2019.
- [Tian19] Tian, Zhi, et al. "Fcos: Fully convolutional one-stage object detection." Proceedings of the IEEE/CVF international conference on computer vision. 2019.
- [Tommasini19] R. Tommasini, D. Calvaresi, and J.-P. Calbimonte. Stream reasoning agents: Bluesky ideas track. In AAMAS, pages 1664–1680, 2019
- [VanAssche21] Van Assche, Dylan, et al. "Leveraging Web of Things W3C recommendations for knowledge graphs generation", Proceedings of the 21st International Conference on Web Engineering, 2021.
- [Vrandečić14] Vrandečić, Denny, and Markus Krötzsch. "Wikidata: a free collaborative knowledgebase." Communications of the ACM 57.10 (2014): 78-85.
- [W3C24] "Web of Things" [Online], <https://www.w3.org/WoT/documentation/>, Retrieved 02/2024
- [Xu17] Xu, Danfei, et al. "Scene graph generation by iterative message passing." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.
- [Zellers18] Zellers, Rowan, et al. "Neural motifs: Scene graph parsing with global context." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.
- [Zhang22] Zhang, Hao, et al. "Dino: Detr with improved denoising anchor boxes for end-to-end object detection." arXiv preprint arXiv:2203.03605 (2022).
- [ZhangJi18] Zhang, Ji, et al. "An interpretable model for scene graph generation." arXiv preprint arXiv:1811.09543 (2018).
- [Zheng23] Zheng, Changgang et al. "DINC: toward distributed in-network computing", ACM CoNEXT and Proceedings of the ACM on Networking (PACMNET), December 2023.
- [Zong23] Zong, Zhuofan, Guanglu Song, and Yu Liu. "Detrs with collaborative hybrid assignments training." Proceedings of the IEEE/CVF international conference on computer vision. 2023.

[Lin14] Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., ... & Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V 13* (pp. 740-755). Springer International Publishing.

[Chen21] Chen, Q., Wang, W., Huang, K., De, S., Coenen, F.: Multi-modal generative adversarial networks for traffic event detection in smart cities. *Expert Systems with Applications* 177, 114,939 (2021).

[Gao20] Gao, J., Li, P., Chen, Z., Zhang, J.: A survey on deep learning for multimodal data fusion. *Neural Computation* 32(5), 829–864 (2020)

[Khadanga19] Khadanga, S., Aggarwal, K., Joty, S., Srivastava, J.: Using clinical notes with time series data for icu management. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 6432–6437 (2019)

[Yang 21] Yang, H., Kuang, L., Xia, F.: Multimodal temporal- clinical note network for mortality prediction. *Journal of Biomedical Semantics* 12(1), 1–14 (2021)