

Future Generation Computer Systems

DiSIF: Distributed Semantic Information Fusion Framework for Smart City Applications

--Manuscript Draft--

Manuscript Number:	FGCS-D-24-02793
Article Type:	Full Length Article
Keywords:	semantic JDL, data fusion, fog computing, stream processing, RSP agent.
Corresponding Author:	Mohsen Kahani, PhD Ferdowsi University of Mashhad Department of Computer Engineering Mashhad, Khorasan Razavi IRAN, ISLAMIC REPUBLIC OF
First Author:	Ramin Rezvani khorashadizadeh, Ph.D candidate
Order of Authors:	Ramin Rezvani khorashadizadeh, Ph.D candidate Mohsen Kahani, Ph.D
Abstract:	<p>In the past decade, the advancement of the Internet of Things (IoT) has facilitated the seamless implementation of integrated systems, generating continuous data streams. This data, characterized by high speed, volume, and heterogeneity, poses significant challenges. To address these challenges, an integrated model for heterogeneous data streams is essential for decision-making in many applications, including smart cities. The use of machine-readable semantic data models in stream reasoning and RDF stream processing is crucial for managing sensor data diversity. The JDL fusion model, particularly its semantic version, enables the integration of heterogeneous data using Semantic Web technologies. However, centralized fusion models are inefficient for the vast data volumes generated in smart cities, leading to network inefficiencies and privacy concerns. User queries, requiring data concept conversions, also suffer from increased execution times in centralized systems. The proposed Distributed Semantic Information Fusion Architecture (called DiSIF) addresses these issues using a three-layer architecture of edge, fog, and cloud, with worker and master nodes at each layer. This model performs low-level processing at the edge, sending only results to higher layers, thus enhancing network efficiency. Horizontal fusion integrates heterogeneous data to tackle sensor data diversity, while vertical fusion manages homogeneous data to reduce the transmission volume. The distributed approach ensures local data processing, while maintaining data privacy. Evaluations comparing the distributed and centralized JDL models depict that the DiSIF framework reduces network load, query execution time, and memory consumption, demonstrating its superiority for smart city applications.</p>
Suggested Reviewers:	<p>Jose Mora j.mora@upm.es the author of StreamQR method</p> <p>Riccardo Tommasini rictomm@gmail.com He is working on similar topics with author. He is Working on Semantic Web, AI, Stream Processing.</p>

Ramin Rezvani

Department of Computer Engineering
Ferdowsi University of Mashhad
Mashhad, Iran
ra.rezvanikhorashadizadeh@mail.um.ac.ir
22 Sep 2024

Dear Editor-in-Chief,

I am pleased to submit our manuscript entitled “**DiSIF: Distributed Semantic Information Fusion Framework for Smart City Applications**” for consideration for publication in *The Future Generation Computer Systems*. This manuscript presents innovative approaches in distributed systems, specifically focusing on a novel **Distributed Semantic JDL Fusion Model** across the Edge, Fog, and Cloud layers, offering significant improvements in decision-making speed and scalability.

The primary contributions of this work include:

1. **Introducing a Distributed Semantic JDL Model** that optimizes decision-making and data fusion across multiple layers (Edge, Fog, and Cloud) presenting a distributed version of the semantic JDL fusion model.
2. **Novel Horizontal and Vertical Query Decomposition** techniques using ontologies to enable parallel execution and concept generation for efficient data fusion.
3. **A Master-Slave Infrastructure** designed to enhance query processing and execution in a three-layer architecture, optimizing interactions between nodes.

Our work is aligned with the scope of *The Future Generation Computer Systems*, particularly in its focus on advances in **distributed systems, Big Data processing, collaborative environments, and high-performance computing**. We believe the manuscript will contribute to the journal’s mission by providing new insights into the design and execution of distributed infrastructures and their application in smart city environments.

We confirm that this manuscript has not been published previously and is not under consideration for publication elsewhere. We also confirm that all authors have approved the content of the manuscript and agree with its submission to *The Future Generation Computer Systems*. Should you require any further information, please do not hesitate to contact me at ra.rezvanikhorashadizadeh@mail.um.ac.ir.

Thank you for considering our submission. We look forward to your feedback.

Sincerely,
Ramin Rezvani
Department of Computer Engineering

Ferdowsi University of Mashhad
Mashhad, Iran
ra.rezvanikhorashadizadeh@mail.um.ac.ir

Author Contributions

- **[Ramin Rezvani-Khorashadizadeh]:**
Conceptualization, Data Curation, Formal Analysis, Investigation, Methodology, Resources, Software, Validation , Visualization Writing – Original Draft.
- **[Mohsen Kahani]:**
Conceptualization, Methodology, Supervision, Writing – Review & Editing.

CRedit Roles Explained:

- **Conceptualization:** Ideas; formulation or evolution of overarching research goals and aims.
- **Methodology:** Development or design of methodology; creation of models.
- **Software:** Programming, software development; designing computer programs; implementation of the computer code and supporting algorithms; testing of existing code components.
- **Data Curation:** Management activities to annotate (produce metadata), scrub data, and maintain research data (including software code, where it is necessary for interpreting the data itself) for initial use and later reuse.
- **Investigation:** Conducting a research and investigation process, specifically performing the experiments, or data/evidence collection.
- **Formal Analysis:** Application of statistical, mathematical, computational, or other formal techniques to analyze or synthesize study data.
- **Writing – Original Draft:** Preparation, creation and/or presentation of the published work, specifically writing the initial draft (including substantive translation).
- **Writing – Review & Editing:** Preparation, creation, and/or presentation of the published work by those from the original research group, specifically critical review, commentary, or revision – including pre- or post-publication stages.
- **Visualization:** Preparation, creation, and/or presentation of the published work, specifically visualization/data presentation.
- **Validation:** Verification, whether as a part of the activity or separate, of the overall replication/reproducibility of results/experiments and other research outputs.
- **Resources:** Provision of study materials, reagents, materials, patients, laboratory samples, animals, instrumentation, computing resources, or other analysis tools.
- **Funding Acquisition:** Acquisition of the financial support for the project leading to this publication.
- **Project Administration:** Management and coordination responsibility for the research activity planning and execution.

Highlights:

1. **Distributed Semantic JDL Fusion Framework:** Introduces a distributed version of the JDL model, optimizing decision-making and data fusion across edge, fog, and cloud layers, ensuring faster and more efficient processing for smart city applications.
2. **Horizontal and Vertical Data Fusion:** Introduced horizontal fusion (combining data from different concepts) and vertical fusion (combining data of the same concept at different scales) for more robust decision-making.
3. **Efficient Distributed Query Execution:** Breaks down independent and complex queries into sub-queries, executing them in parallel across worker and master nodes, significantly reducing query execution time and optimizing memory and network bandwidth usage.
4. **Reduced Network Load:** By processing data locally at the edge, only the processed results are transmitted to higher layers, minimizing network bandwidth utilization and enhancing overall network efficiency.
5. **Enhanced Data Privacy:** Maintains privacy by processing data locally and avoiding the transmission of raw data across the network, addressing privacy concerns commonly found in centralized systems.
6. **Support for Dynamic and Complex Queries:** The framework efficiently handles the dynamic execution of complex queries introduced during runtime, outperforming previous models focused on predefined queries.

```
This is pdfTeX, Version 3.141592653-2.6-1.40.25 (TeX Live 2023)
(preloaded format=pdflatex 2024.3.8)  23 SEP 2024 05:17
entering extended mode
  restricted \writel8 enabled.
  %&-line parsing enabled.
**disif_article.tex
(./DiSIF_Article.tex
LaTeX2e <2023-11-01> patch level 1
L3 programming layer <2024-02-20>
(c:/texlive/2023/texmf-dist/tex/latex/elsarticle/elsarticle.cls
Document Class: elsarticle 2020/11/20, 3.3: Elsevier Ltd
(c:/texlive/2023/texmf-dist/tex/latex/l3kernel/expl3.sty
Package: expl3 2024-02-20 L3 programming layer (loader)
(c:/texlive/2023/texmf-dist/tex/latex/l3backend/l3backend-pdftex.def
File: l3backend-pdftex.def 2024-02-20 L3 backend support: PDF output
(pdfTeX)
\l__color_backend_stack_int=\count188
\l__pdf_internal_box=\box51
)) (c:/texlive/2023/texmf-dist/tex/latex/l3packages/xparse/xparse.sty
Package: xparse 2024-02-18 L3 Experimental document command parser
) (c:/texlive/2023/texmf-dist/tex/latex/etoolbox/etoolbox.sty
Package: etoolbox 2020/10/05 v2.5k e-TeX tools for LaTeX (JAW)
\etb@tempcnta=\count189
)
\@bls=\dimen140
(c:/texlive/2023/texmf-dist/tex/latex/base/article.cls
Document Class: article 2023/05/17 v1.4n Standard LaTeX document class
(c:/texlive/2023/texmf-dist/tex/latex/base/size10.clo
File: size10.clo 2023/05/17 v1.4n Standard LaTeX file (size option)
)
\c@part=\count190
\c@section=\count191
\c@subsection=\count192
\c@subsubsection=\count193
\c@paragraph=\count194
\c@subparagraph=\count195
\c@figure=\count196
\c@table=\count197
\abovecaptionskip=\skip48
\belowcaptionskip=\skip49
\bibindent=\dimen141
) (c:/texlive/2023/texmf-dist/tex/latex/graphics/graphicx.sty
Package: graphicx 2021/09/16 v1.2d Enhanced LaTeX Graphics (DPC,SPQR)
(c:/texlive/2023/texmf-dist/tex/latex/graphics/keyval.sty
Package: keyval 2022/05/29 v1.15 key=value parser (DPC)
\KV@toks@=\toks17
) (c:/texlive/2023/texmf-dist/tex/latex/graphics/graphics.sty
Package: graphics 2022/03/10 v1.4e Standard LaTeX Graphics (DPC,SPQR)
(c:/texlive/2023/texmf-dist/tex/latex/graphics/trig.sty
Package: trig 2021/08/11 v1.11 sin cos tan (DPC)
) (c:/texlive/2023/texmf-dist/tex/latex/graphics-cfg/graphics.cfg
File: graphics.cfg 2016/06/04 v1.11 sample graphics configuration
)
Package graphics Info: Driver file: pdftex.def on input line 107.
```

```

(c:/texlive/2023/texmf-dist/tex/latex/graphics-def/pdftex.def
File: pdftex.def 2022/09/22 v1.2b Graphics/color driver for pdftex
))
\Gin@req@height=\dimen142
\Gin@req@width=\dimen143
)
\c@tnote=\count198
\c@fnote=\count199
\c@cnote=\count266
\c@ead=\count267
\c@author=\count268
\@eadauthor=\toks18
\c@affn=\count269
\absbox=\box52
\elsarticlehighlightsbox=\box53
\elsarticlegrabsbox=\box54
\keybox=\box55
\Columnwidth=\dimen144
\space@left=\dimen145
\els@boxa=\box56
\els@boxb=\box57
\leftMargin=\dimen146
\@enLab=\toks19
\@sep=\skip50
\@@sep=\skip51
(./DiSIF_Article.spl) (c:/texlive/2023/texmf-
dist/tex/latex/natbib/natbib.sty
Package: natbib 2010/09/13 8.31b (PWD, AO)
\bibhang=\skip52
\bibsep=\skip53
LaTeX Info: Redefining \cite on input line 694.
\c@NAT@ctr=\count270
)
\splwrite=\write3
\openout3 = `DiSIF_Article.spl'.

```

```

(c:/texlive/2023/texmf-dist/tex/latex/geometry/geometry.sty
Package: geometry 2020/01/02 v5.9 Page Geometry
(c:/texlive/2023/texmf-dist/tex/generic/iftex/ifvtex.sty
Package: ifvtex 2019/10/25 v1.7 ifvtex legacy package. Use iftex instead.
(c:/texlive/2023/texmf-dist/tex/generic/iftex/iftex.sty
Package: iftex 2022/02/03 v1.0f TeX engine tests
))
\Gm@cnth=\count271
\Gm@cntv=\count272
\c@Gm@tempcnt=\count273
\Gm@bindingoffset=\dimen147
\Gm@wd@mp=\dimen148
\Gm@odd@mp=\dimen149
\Gm@even@mp=\dimen150
\Gm@layoutwidth=\dimen151
\Gm@layoutheight=\dimen152
\Gm@layouthoffset=\dimen153
\Gm@layoutvoffset=\dimen154

```

```

\Gm@dimlist=\toks20
) (c:/texlive/2023/texmf-dist/tex/latex/base/fleqn.clo
File: fleqn.clo 2016/12/29 v1.2b Standard LaTeX option (flush left
equations)
\mathindent=\skip54
Applying: [2015/01/01] Make \[ robust on input line 50.
LaTeX Info: Redefining \[ on input line 51.
Already applied: [0000/00/00] Make \[ robust on input line 62.
Applying: [2015/01/01] Make \] robust on input line 74.
LaTeX Info: Redefining \] on input line 75.
Already applied: [0000/00/00] Make \] robust on input line 83.
)
\appnamewidth=\dimen155
) (c:/texlive/2023/texmf-dist/tex/latex/lineno/lineno.sty
Package: lineno 2023/05/20 line numbers on paragraphs v5.3
(c:/texlive/2023/texmf-dist/tex/latex/kvoptions/kvoptions.sty
Package: kvoptions 2022-06-15 v3.15 Key value format for package options
(HO)
(c:/texlive/2023/texmf-dist/tex/generic/ltxcmds/ltxcmds.sty
Package: ltxcmds 2023-12-04 v1.26 LaTeX kernel commands for general use
(HO)
) (c:/texlive/2023/texmf-dist/tex/latex/kvsetkeys/kvsetkeys.sty
Package: kvsetkeys 2022-10-05 v1.19 Key value parser (HO)
))
\linenopenalty=\count274
\output=\toks21
\linenoprevgraf=\count275
\linenumbersep=\dimen156
\linenumberwidth=\dimen157
\c@linenumber=\count276
\c@pagewiselinenumber=\count277
\c@LN@truepage=\count278
\c@internallinenumber=\count279
\c@internallinenumbers=\count280
\quotelinenumbersep=\dimen158
\bframerule=\dimen159
\bframesep=\dimen160
\bframebox=\box58
LaTeX Info: Redefining \ on input line 3180.
) (c:/texlive/2023/texmf-dist/tex/latex/amsfonts/amssymb.sty
Package: amssymb 2013/01/14 v3.01 AMS font symbols
(c:/texlive/2023/texmf-dist/tex/latex/amsfonts/amsfonts.sty
Package: amsfonts 2013/01/14 v3.01 Basic AMSFonts support
\@emptytoks=\toks22
\symAMSA=\mathgroup4
\symAMSb=\mathgroup5
LaTeX Font Info: Redefining math symbol \hbar on input line 98.
LaTeX Font Info: Overwriting math alphabet '\mathfrak' in version
'bold'
(Font) U/euf/m/n --> U/euf/b/n on input line 106.
)) (c:/texlive/2023/texmf-dist/tex/latex/caption/subcaption.sty
Package: subcaption 2023/07/28 v1.6b Sub-captions (AR)
(c:/texlive/2023/texmf-dist/tex/latex/caption/caption.sty
Package: caption 2023/08/05 v3.6o Customizing captions (AR)

```



```

(c:/texlive/2023/texmf-dist/tex/latex/caption/caption3.sty
Package: caption3 2023/07/31 v2.4d caption3 kernel (AR)
\caption@tempdima=\dimen161
\captionmargin=\dimen162
\caption@leftmargin=\dimen163
\caption@rightmargin=\dimen164
\caption@width=\dimen165
\caption@indent=\dimen166
\caption@parindent=\dimen167
\caption@hangindent=\dimen168
Package caption Info: elsarticle document class detected.
(c:/texlive/2023/texmf-dist/tex/latex/caption/caption-elsarticle.sto
File: caption-elsarticle.sto 2020/08/22 v2.0 Adaption of the caption
package to
  the elsarticle document class (AR)
))
\c@caption@flags=\count281
\c@continuedfloat=\count282
)
Package caption Info: New subtype `subfigure' on input line 238.
\c@subfigure=\count283
Package caption Info: New subtype `subtable' on input line 238.
\c@subtable=\count284
) (c:/texlive/2023/texmf-dist/tex/latex/algorithms/algorithm.sty
Package: algorithm 2009/08/24 v0.1 Document Style `algorithm' - floating
enviro
nment
(c:/texlive/2023/texmf-dist/tex/latex/float/float.sty
Package: float 2001/11/08 v1.3d Float enhancements (AL)
\c@float@type=\count285
\float@exts=\toks23
\float@box=\box59
\@float@everytoks=\toks24
\@floatcapt=\box60
) (c:/texlive/2023/texmf-dist/tex/latex/base/ifthen.sty
Package: ifthen 2022/04/13 v1.1d Standard LaTeX ifthen package (DPC)
)
\@float@every@algorithm=\toks25
\c@algorithm=\count286
) (c:/texlive/2023/texmf-dist/tex/latex/algorithmicx/algpseudocode.sty
Package: algpseudocode
(c:/texlive/2023/texmf-dist/tex/latex/algorithmicx/algorithmicx.sty
Package: algorithmicx 2005/04/27 v1.2 Algorithmicx
Document Style algorithmicx 1.2 - a greatly improved `algorithmic' style
\c@ALG@line=\count287
\c@ALG@rem=\count288
\c@ALG@nested=\count289
\ALG@tlm=\skip55
\ALG@thistlm=\skip56
\c@ALG@Lnr=\count290
\c@ALG@blocknr=\count291
\c@ALG@storecount=\count292
\c@ALG@tmpcounter=\count293
\ALG@tmplength=\skip57

```

```

)
Document Style - pseudocode environments for use with the `algorithmicx'
style
) (c:/texlive/2023/texmf-dist/tex/latex/enumitem/enumitem.sty
Package: enumitem 2019/06/20 v3.9 Customized lists
\labelindent=\skip58
\enit@outerparindent=\dimen169
\enit@toks=\toks26
\enit@inbox=\box61
\enit@count@id=\count294
\enitdp@description=\count295
) (c:/texlive/2023/texmf-dist/tex/latex/amsmath/amsmath.sty
Package: amsmath 2023/05/13 v2.17o AMS math features
\@mathmargin=\skip59
For additional information on amsmath, use the `?' option.
(c:/texlive/2023/texmf-dist/tex/latex/amsmath/amstext.sty
Package: amstext 2021/08/26 v2.01 AMS text
(c:/texlive/2023/texmf-dist/tex/latex/amsmath/amsgen.sty
File: amsgen.sty 1999/11/30 v2.0 generic functions
\@emptytoks=\toks27
\ex@=\dimen170
)) (c:/texlive/2023/texmf-dist/tex/latex/amsmath/amsbsy.sty
Package: amsbsy 1999/11/29 v1.2d Bold Symbols
\pmbraise@=\dimen171
) (c:/texlive/2023/texmf-dist/tex/latex/amsmath/amsopn.sty
Package: amsopn 2022/04/08 v2.04 operator names
)
\inf@bad=\count296
LaTeX Info: Redefining \frac on input line 234.
\uproot@=\count297
\leftroot@=\count298
LaTeX Info: Redefining \overline on input line 399.
LaTeX Info: Redefining \colon on input line 410.
\classnum@=\count299
\DOTSCASE@=\count300
LaTeX Info: Redefining \ldots on input line 496.
LaTeX Info: Redefining \dots on input line 499.
LaTeX Info: Redefining \cdots on input line 620.
\Mathstrutbox@=\box62
\strutbox@=\box63
LaTeX Info: Redefining \big on input line 722.
LaTeX Info: Redefining \Big on input line 723.
LaTeX Info: Redefining \bigg on input line 724.
LaTeX Info: Redefining \Bigg on input line 725.
\big@size=\dimen172
LaTeX Font Info: Redefining font encoding OML on input line 743.
LaTeX Font Info: Redefining font encoding OMS on input line 744.
\maccc@depth=\count301
LaTeX Info: Redefining \bmod on input line 905.
LaTeX Info: Redefining \pmod on input line 910.
LaTeX Info: Redefining \smash on input line 940.
LaTeX Info: Redefining \relbar on input line 970.
LaTeX Info: Redefining \Relbar on input line 971.
\c@MaxMatrixCols=\count302

```

```

\dotsspace@=\muskip16
\c@parentequation=\count303
\dspbrk@lvl=\count304
\tag@help=\toks28
\row@=\count305
\column@=\count306
\maxfields@=\count307
\andhelp@=\toks29
\eqnshift@=\dimen173
\alignsep@=\dimen174
\tagshift@=\dimen175
\tagwidth@=\dimen176
\totwidth@=\dimen177
\lineht@=\dimen178
\@envbody=\toks30
\multlinegap=\skip60
\multlinetaggap=\skip61
\mathdisplay@stack=\toks31
LaTeX Info: Redefining \[ on input line 2953.
LaTeX Info: Redefining \] on input line 2954.
)
\linenoamsmath@ams@eqpen=\count308
(c:/texlive/2023/texmf-dist/tex/latex/svg/svg.sty
Package: svg 2020/11/26 v2.02k (include SVG pictures)
(c:/texlive/2023/texmf-dist/tex/latex/koma-script/scrbase.sty
Package: scrbase 2023/07/07 v3.41 KOMA-Script package (KOMA-Script-
independent
basics and keyval usage)
(c:/texlive/2023/texmf-dist/tex/latex/koma-script/scrlfile.sty
Package: scrlfile 2023/07/07 v3.41 KOMA-Script package (file load hooks)
(c:/texlive/2023/texmf-dist/tex/latex/koma-script/scrlfile-hook.sty
Package: scrlfile-hook 2023/07/07 v3.41 KOMA-Script package (using LaTeX
hooks)

(c:/texlive/2023/texmf-dist/tex/latex/koma-script/scrlogo.sty
Package: scrlogo 2023/07/07 v3.41 KOMA-Script package (logo)
)))
Applying: [2021/05/01] Usage of raw or classic option list on input line
252.
Already applied: [0000/00/00] Usage of raw or classic option list on
input line
368.
) (c:/texlive/2023/texmf-dist/tex/generic/pdftexcmds/pdftexcmds.sty
Package: pdftexcmds 2020-06-27 v0.33 Utility functions of pdfTeX for
LuaTeX (HO
)
(c:/texlive/2023/texmf-dist/tex/generic/infwarerr/infwarerr.sty
Package: infwarerr 2019/12/03 v1.5 Providing info/warning/error messages
(HO)
)
Package pdftexcmds Info: \pdf@primitive is available.
Package pdftexcmds Info: \pdf@ifprimitive is available.
Package pdftexcmds Info: \pdfdraftmode found.
) (c:/texlive/2023/texmf-dist/tex/latex/trimspaces/trimspaces.sty

```

```

Package: trimspaces 2009/09/17 v1.1 Trim spaces around a token list
) (c:/texlive/2023/texmf-dist/tex/latex/tools/shellessc.sty
Package: shellessc 2023/07/08 v1.0d unified shell escape interface for
LaTeX
Package shellessc Info: Restricted shell escape enabled on input line 77.
)
\c@svg@param@lastpage=\count309
\svg@box=\box64
\c@svg@param@currpage=\count310
) (c:/texlive/2023/texmf-dist/tex/latex/xcolor/xcolor.sty
Package: xcolor 2023/11/15 v3.01 LaTeX color extensions (UK)
(c:/texlive/2023/texmf-dist/tex/latex/graphics-cfg/color.cfg
File: color.cfg 2016/01/02 v1.6 sample color configuration
)
Package xcolor Info: Driver file: pdftex.def on input line 274.
(c:/texlive/2023/texmf-dist/tex/latex/graphics/mathcolor.ltx)
Package xcolor Info: Model `cmy' substituted by `cmy0' on input line
1350.
Package xcolor Info: Model `hsb' substituted by `rgb' on input line 1354.
Package xcolor Info: Model `RGB' extended on input line 1366.
Package xcolor Info: Model `HTML' substituted by `rgb' on input line
1368.
Package xcolor Info: Model `Hsb' substituted by `hsb' on input line 1369.
Package xcolor Info: Model `tHsb' substituted by `hsb' on input line
1370.
Package xcolor Info: Model `HSB' substituted by `hsb' on input line 1371.
Package xcolor Info: Model `Gray' substituted by `gray' on input line
1372.
Package xcolor Info: Model `wave' substituted by `hsb' on input line
1373.
) (c:/texlive/2023/texmf-dist/tex/latex/transparent/transparent.sty
Package: transparent 2022-10-27 v1.5 Transparency with color stacks
(c:/texlive/2023/texmf-dist/tex/latex/transparent/transparent-
nometadata.sty
Package: transparent-nometadata 2022-10-27 v1.5 Transparency via pdfTeX's
color
stack (H0)
(c:/texlive/2023/texmf-dist/tex/latex/auxhook/auxhook.sty
Package: auxhook 2019-12-17 v1.6 Hooks for auxiliary files (H0)
))) (c:/texlive/2023/texmf-dist/tex/latex/pgf/systemlayer/pgfsys.sty
(c:/texliv
e/2023/texmf-dist/tex/latex/pgf/utilities/pgfrcs.sty
(c:/texlive/2023/texmf-dis
t/tex/generic/pgf/utilities/pgfutil-common.tex
\pgfutil@everybye=\toks32
\pgfutil@tempdima=\dimen179
\pgfutil@tempdimb=\dimen180
) (c:/texlive/2023/texmf-dist/tex/generic/pgf/utilities/pgfutil-latex.def
\pgfutil@abb=\box65
) (c:/texlive/2023/texmf-dist/tex/generic/pgf/utilities/pgfrcs.code.tex
(c:/tex
live/2023/texmf-dist/tex/generic/pgf/pgf.revision.tex)
Package: pgfrcs 2023-01-15 v3.1.10 (3.1.10)

```

```

)) (c:/texlive/2023/texmf-
dist/tex/generic/pgf/systemlayer/pgfsys.code.tex
Package: pgfsys 2023-01-15 v3.1.10 (3.1.10)
(c:/texlive/2023/texmf-dist/tex/generic/pgf/utilities/pgfkeys.code.tex
\pgfkeys@pathtoks=\toks33
\pgfkeys@temptoks=\toks34

(c:/texlive/2023/texmf-
dist/tex/generic/pgf/utilities/pgfkeyslibraryfiltered.co
de.tex
\pgfkeys@tmptoks=\toks35
))
\pgf@x=\dimen181
\pgf@y=\dimen182
\pgf@xa=\dimen183
\pgf@ya=\dimen184
\pgf@xb=\dimen185
\pgf@yb=\dimen186
\pgf@xc=\dimen187
\pgf@yc=\dimen188
\pgf@xd=\dimen189
\pgf@yd=\dimen190
\w@pgf@writea=\write4
\r@pgf@reada=\read2
\c@pgf@counta=\count311
\c@pgf@countb=\count312
\c@pgf@countc=\count313
\c@pgf@countd=\count314
\t@pgf@toka=\toks36
\t@pgf@tokb=\toks37
\t@pgf@tokc=\toks38
\pgf@sys@id@count=\count315
(c:/texlive/2023/texmf-dist/tex/generic/pgf/systemlayer/pgf.cfg
File: pgf.cfg 2023-01-15 v3.1.10 (3.1.10)
)
Driver file for pgf: pgfsys-pdftex.def
(c:/texlive/2023/texmf-dist/tex/generic/pgf/systemlayer/pgfsys-pdftex.def
File: pgfsys-pdftex.def 2023-01-15 v3.1.10 (3.1.10)
(c:/texlive/2023/texmf-dist/tex/generic/pgf/systemlayer/pgfsys-common-
pdf.def
File: pgfsys-common-pdf.def 2023-01-15 v3.1.10 (3.1.10)
)))
(c:/texlive/2023/texmf-
dist/tex/generic/pgf/systemlayer/pgfsyssoftpath.code.tex
File: pgfsyssoftpath.code.tex 2023-01-15 v3.1.10 (3.1.10)
\pgfsyssoftpath@smallbuffer@items=\count316
\pgfsyssoftpath@bigbuffer@items=\count317
)
(c:/texlive/2023/texmf-
dist/tex/generic/pgf/systemlayer/pgfsysprotocol.code.tex
File: pgfsysprotocol.code.tex 2023-01-15 v3.1.10 (3.1.10)
)) (c:/texlive/2023/texmf-dist/tex/latex/psnfss/pifont.sty
Package: pifont 2020/03/25 PSNFSS-v9.3 Pi font support (SPQR)

```

LaTeX Font Info: Trying to load font information for U+pzd on input line 63.

(c:/texlive/2023/texmf-dist/tex/latex/psnfss/upzd.fd
File: upzd.fd 2001/06/04 font definitions for U/pzd.

)
LaTeX Font Info: Trying to load font information for U+psy on input line 64.

(c:/texlive/2023/texmf-dist/tex/latex/psnfss/upsy.fd
File: upsy.fd 2001/06/04 font definitions for U/psy.

)) (c:/texlive/2023/texmf-dist/tex/latex/setspace/setspace.sty

Package: setspace 2022/12/04 v6.7b set line spacing

) (c:/texlive/2023/texmf-dist/tex/latex/booktabs/booktabs.sty

Package: booktabs 2020/01/12 v1.61803398 Publication quality tables

\heavyrulewidth=\dimen191

\lightrulewidth=\dimen192

\cmidrulewidth=\dimen193

\belowrulesep=\dimen194

\belowbottomsep=\dimen195

\aboverulesep=\dimen196

\abovetopsep=\dimen197

\cmidrulesep=\dimen198

\cmidrulekern=\dimen199

\defaultaddspace=\dimen256

\@cmidla=\count318

\@cmidlb=\count319

\@aboverulesep=\dimen257

\@belowrulesep=\dimen258

\@thisruleclass=\count320

\@lastruleclass=\count321

\@thisrulewidth=\dimen259

) (c:/texlive/2023/texmf-dist/tex/latex/adjustbox/adjustbox.sty

Package: adjustbox 2022/10/17 v1.3a Adjusting TeX boxes (trim, clip, ...)

(c:/texlive/2023/texmf-dist/tex/latex/xkeyval/xkeyval.sty

Package: xkeyval 2022/06/16 v2.9 package option processing (HA)

(c:/texlive/2023/texmf-dist/tex/generic/xkeyval/xkeyval.tex

(c:/texlive/2023/te

xmf-dist/tex/generic/xkeyval/xkvutils.tex

\XKV@toks=\toks39

\XKV@tempa@toks=\toks40

)

\XKV@depth=\count322

File: xkeyval.tex 2014/12/03 v2.7a key=value parser (HA)

)) (c:/texlive/2023/texmf-dist/tex/latex/adjustbox/adjcalc.sty

Package: adjcalc 2012/05/16 v1.1 Provides advanced setlength with multiple back

-ends (calc, etex, pgfmath)

) (c:/texlive/2023/texmf-dist/tex/latex/adjustbox/trimclip.sty

Package: trimclip 2020/08/19 v1.2 Trim and clip general TeX material

(c:/texlive/2023/texmf-dist/tex/latex/collectbox/collectbox.sty

Package: collectbox 2022/10/17 v0.4c Collect macro arguments as boxes

\collectedbox=\box66

)

```

\tc@llx=\dimen260
\tc@lly=\dimen261
\tc@urx=\dimen262
\tc@ury=\dimen263
Package trimclip Info: Using driver 'tc-pdftex.def'.
(c:/texlive/2023/texmf-dist/tex/latex/adjustbox/tc-pdftex.def
File: tc-pdftex.def 2019/01/04 v2.2 Clipping driver for pdftex
))
\adjbox@Width=\dimen264
\adjbox@Height=\dimen265
\adjbox@Depth=\dimen266
\adjbox@Totalheight=\dimen267
\adjbox@pwidth=\dimen268
\adjbox@pheight=\dimen269
\adjbox@pdepth=\dimen270
\adjbox@ptotalheight=\dimen271
(c:/texlive/2023/texmf-dist/tex/latex/ifoddpaper/ifoddpaper.sty
Package: ifoddpaper 2022/10/18 v1.2 Conditionals for odd/even page
detection
\c@checkoddpaper=\count323
) (c:/texlive/2023/texmf-dist/tex/latex/varwidth/varwidth.sty
Package: varwidth 2009/03/30 ver 0.92; Variable-width minipages
\@vwid@box=\box67
\sift@deathcycles=\count324
\@vwid@loff=\dimen272
\@vwid@roff=\dimen273
)) (./DiSIF_Article.aux)
\openout1 = `DiSIF_Article.aux'.

```

```

LaTeX Font Info: Checking defaults for OML/cmm/m/it on input line 29.
LaTeX Font Info: ... okay on input line 29.
LaTeX Font Info: Checking defaults for OMS/cmsy/m/n on input line 29.
LaTeX Font Info: ... okay on input line 29.
LaTeX Font Info: Checking defaults for OT1/cmr/m/n on input line 29.
LaTeX Font Info: ... okay on input line 29.
LaTeX Font Info: Checking defaults for T1/cmr/m/n on input line 29.
LaTeX Font Info: ... okay on input line 29.
LaTeX Font Info: Checking defaults for TS1/cmr/m/n on input line 29.
LaTeX Font Info: ... okay on input line 29.
LaTeX Font Info: Checking defaults for OMX/cmex/m/n on input line 29.
LaTeX Font Info: ... okay on input line 29.
LaTeX Font Info: Checking defaults for U/cmr/m/n on input line 29.
LaTeX Font Info: ... okay on input line 29.
(c:/texlive/2023/texmf-dist/tex/context/base/mkii/supp-pdf.mkii
[Loading MPS to PDF converter (version 2006.09.02).]
\scratchcounter=\count325
\scratchdimen=\dimen274
\scratchbox=\box68
\nofMPsegments=\count326
\nofMParguments=\count327
\everyMPshowfont=\toks41
\MPscratchCnt=\count328
\MPscratchDim=\dimen275
\MPnumerator=\count329

```

```

\makeMPintoPDFobject=\count330
\everyMPtoPDFconversion=\toks42
) (c:/texlive/2023/texmf-dist/tex/latex/epstopdf-pkg/epstopdf-base.sty
Package: epstopdf-base 2020-01-24 v2.11 Base part for package epstopdf
Package epstopdf-base Info: Redefining graphics rule for '.eps' on input
line 4
85.
(c:/texlive/2023/texmf-dist/tex/latex/latexconfig/epstopdf-sys.cfg
File: epstopdf-sys.cfg 2010/07/13 v1.3 Configuration of (r)epstopdf for
TeX Liv
e
))
*geometry* driver: auto-detecting
*geometry* detected driver: pdftex
*geometry* verbose mode - [ preamble ] result:
* driver: pdftex
* paper: custom
* layout: <same size as paper>
* layoutoffset: (h,v)=(0.0pt,0.0pt)
* hratio: 1:1
* vratio: 1:1
* modes:
* h-part: (L,W,R)=(37.75394pt, 522.0pt, 37.75394pt)
* v-part: (T,H,B)=(81.52342pt, 682.0pt, 81.52342pt)
* \paperwidth=597.50787pt
* \paperheight=845.04684pt
* \textwidth=522.0pt
* \textheight=682.0pt
* \oddsidemargin=-34.51605pt
* \evensidemargin=-34.51605pt
* \topmargin=-52.74657pt
* \headheight=50.0pt
* \headsep=12.0pt
* \topskip=10.0pt
* \footskip=18.0pt
* \marginparwidth=4.0pt
* \marginparsep=10.0pt
* \columnsep=18.0pt
* \skip\footins=24.0pt plus 2.0pt minus 12.0pt
* \hoffset=0.0pt
* \voffset=0.0pt
* \mag=1000
* \@twocolumntrue
* \@twosidefalse
* \@mparswitchfalse
* \@reversemarginfalse
* (lin=72.27pt=25.4mm, 1cm=28.453pt)

Package caption Info: Begin \AtBeginDocument code.
Package caption Info: float package is loaded.
Package caption Info: End \AtBeginDocument code.
LaTeX Font Info: Trying to load font information for U+msa on input
line 59.

```



```
(c:/texlive/2023/texmf-dist/tex/latex/amsfonts/umsa.fd
File: umsa.fd 2013/01/14 v3.01 AMS symbols A
)
LaTeX Font Info:    Trying to load font information for U+msb on input
line 59.
```

```
(c:/texlive/2023/texmf-dist/tex/latex/amsfonts/umsb.fd
File: umsb.fd 2013/01/14 v3.01 AMS symbols B
)
Underfull \vbox (badness 1038) has occurred while \output is active []
```

```
Overfull \hbox (2.22221pt too wide) has occurred while \output is active
[]
[]
```

```
[1{c:/texlive/2023/texmf-var/fonts/map/pdftex/updmap/pdftex.map}
```

```
] [2{c:/texlive/2023/texmf-dist/fonts/enc/dvips/cm-super/cm-super-
ts1.enc}]
Underfull \hbox (badness 4391) in paragraph at lines 127--128
[]\OT1/cmr/bx/n/10 Distributed Se-man-tic JDL Fu-sion Model
[]
```

```
Underfull \hbox (badness 1838) in paragraph at lines 137--138
[]\OT1/cmr/bx/n/10 Introducing a dis-tributed query ex-e-cution
[]
```

```
Underfull \hbox (badness 10000) in paragraph at lines 172--172
[]\OT1/cmr/m/n/10 Object-
[]
```

```
[3]
```

```
Package natbib Warning: Citation `Hall1997' on page 4 undefined on input
line 1
91.
```

```
Package natbib Warning: Citation `noughabi2013semfus' on page 4 undefined
on in
put line 227.
```

```
[4]
```

```
LaTeX Warning: File `C:/ArticleLatexImagesFinal/Semantic_JDL.jpg' not
found on
input line 250.
```

! Package pdftex.def Error: File
`C:/ArticleLatexImagesFinal/Semantic_JDL.jpg'
not found: using draft setting.

See the pdftex.def package documentation for explanation.
Type H <return> for immediate help.

...

l.250 ...ArticleLatexImagesFinal/Semantic_JDL.jpg}

Try typing <return> to proceed.
If that doesn't work, type X <return> to quit.

Package natbib Warning: Citation `noughabi2013semfus' on page 5 undefined
on in
put line 251.

Package natbib Warning: Citation `noughabi2013semfus' on page 5 undefined
on in
put line 251.

[5]

Package natbib Warning: Citation `article28' on page 6 undefined on input
line
270.

Package natbib Warning: Citation `Al-Baltah2020' on page 6 undefined on
input l
ine 274.

Package natbib Warning: Citation `Samadian2014' on page 6 undefined on
input li
ne 274.

Underfull \vbox (badness 10000) has occurred while \output is active []

[6]

Package natbib Warning: Citation `Samadian2014' on page 7 undefined on
input li
ne 278.

Package natbib Warning: Citation `inproceedings31' on page 7 undefined on
input
line 280.

Package natbib Warning: Citation `article32' on page 7 undefined on input line 282.

Package natbib Warning: Citation `article32' on page 7 undefined on input line 282.

Package natbib Warning: Citation `article32' on page 7 undefined on input line 282.

Package natbib Warning: Citation `inproceedings33' on page 7 undefined on input line 284.

Package natbib Warning: Citation `Wang2017' on page 7 undefined on input line 286.

Package natbib Warning: Citation `article35' on page 7 undefined on input line 290.

Underfull \hbox (badness 1735) in paragraph at lines 290--291
[]\OT1/cmr/m/n/10 Another in-tro-duced ar-chi-tec-ture for col-lect-ing
and
[]

Package natbib Warning: Citation `Zafeiropoulos2008' on page 7 undefined on input line 292.

Package natbib Warning: Citation `belcao2020streaming' on page 7 undefined on input line 294.

Package natbib Warning: Citation `zappatore2023semantic' on page 7 undefined on input line 296.

[7]

Package natbib Warning: Citation `GHEISARI20211' on page 8 undefined on input line 1

ine 300.

Package natbib Warning: Citation `de2023spatio' on page 8 undefined on
input line 303.

Package natbib Warning: Citation `Garca2011ExploringTL' on page 8
undefined on
input line 309.

Package natbib Warning: Citation `Ahmed2017BringingCC' on page 8
undefined on i
nput line 311.

Package natbib Warning: Citation `Rahman2019BlockchainAI' on page 8
undefined o
n input line 311.

Package natbib Warning: Citation `Abbas2018MobileEC' on page 8 undefined
on inp
ut line 313.

Package natbib Warning: Citation `Ahmed2017BringingCC' on page 8
undefined on i
nput line 313.

Package natbib Warning: Citation `Perera2017FogCF' on page 8 undefined on
input
line 315.

Package natbib Warning: Citation `Hu2017SurveyOF' on page 8 undefined on
input
line 317.

Package natbib Warning: Citation `Shi2016EdgeCV' on page 8 undefined on
input l
ine 319.

Package natbib Warning: Citation `Baccarelli2017FogOE' on page 8
undefined on i
nput line 321.

Package natbib Warning: Citation `Giordano2016SmartAA' on page 8
undefined on input line 323.

[8]

Package natbib Warning: Citation `inproceedings61' on page 9 undefined on
input line 325.

Package natbib Warning: Citation `TULI201922' on page 9 undefined on
input line 330.

Package natbib Warning: Citation `article63' on page 9 undefined on input
line 332.

Package natbib Warning: Citation `article63' on page 9 undefined on input
line 334.

Package natbib Warning: Citation `Arkian2017' on page 9 undefined on
input line 336.

Package natbib Warning: Citation `Dastjerdi2016' on page 9 undefined on
input line 340.

[9]

Package natbib Warning: Citation `ortiz2022atmosphere' on page 10
undefined on input line 342.

Package natbib Warning: Citation `bonte2023towards' on page 10 undefined
on input line 344.

Package natbib Warning: Citation `XHAF2020730' on page 10 undefined on
input line 347.

Package natbib Warning: Citation `SELLAMI202464' on page 10 undefined on
input

line 350.

Package natbib Warning: Citation `anicic2011ep' on page 10 undefined on
input 1
line 358.

Package natbib Warning: Citation `komazec2012sparkwave' on page 10
undefined on
input line 358.

Package natbib Warning: Citation `rinne2012instans' on page 10 undefined
on inp
ut line 358.

Package natbib Warning: Citation `barbieri2010c' on page 10 undefined on
input
line 358.

Package natbib Warning: Citation `le2011native' on page 10 undefined on
input 1
line 358.

Package natbib Warning: Citation `le2013elastic' on page 10 undefined on
input
line 362.

Package natbib Warning: Citation `ren2018distributed' on page 10
undefined on i
nput line 362.

Package natbib Warning: Citation `ren2017strider' on page 10 undefined on
input
line 362.

Package natbib Warning: Citation `dia2018drss' on page 10 undefined on
input li
ne 362.

Package natbib Warning: Citation `mebrek2020stream' on page 10 undefined
on inp
ut line 364.

Package natbib Warning: Citation `khrouf2016waves' on page 10 undefined on input line 364.

Package natbib Warning: Citation `barbieri2010c' on page 10 undefined on input line 364.

Package natbib Warning: Citation `calbimonte2010enabling' on page 10 undefined on input line 364.

Package natbib Warning: Citation `komazec2012sparkwave' on page 10 undefined on input line 364.

Package natbib Warning: Citation `le2011native' on page 10 undefined on input line 364.

Package natbib Warning: Citation `Zafeiropoulos2008' on page 10 undefined on input line 407.

Package natbib Warning: Citation `inproceedings31' on page 10 undefined on input line 407.

Package natbib Warning: Citation `article32' on page 10 undefined on input line 407.

Package natbib Warning: Citation `article28' on page 10 undefined on input line 407.

Package natbib Warning: Citation `inproceedings33' on page 10 undefined on input line 407.

Package natbib Warning: Citation `article35' on page 10 undefined on input line 407.

Package natbib Warning: Citation `Dastjerdi2016' on page 10 undefined on input line 407.

Package natbib Warning: Citation `Giordano2016SmartAA' on page 10 undefined on input line 407.

Package natbib Warning: Citation `khrouf2016waves' on page 10 undefined on input line 407.

Package natbib Warning: Citation `calbimonte2016query' on page 10 undefined on input line 407.

Package natbib Warning: Citation `Wang2017' on page 10 undefined on input line 407.

Package natbib Warning: Citation `Arkian2017' on page 10 undefined on input line 407.

Package natbib Warning: Citation `dia2018drss' on page 10 undefined on input line 407.

Package natbib Warning: Citation `TULI201922' on page 10 undefined on input line 407.

Package natbib Warning: Citation `article63' on page 10 undefined on input line 407.

Package natbib Warning: Citation `Al-Baltah2020' on page 10 undefined on input line 407.

Package natbib Warning: Citation `mebrek2020stream' on page 10 undefined on input line 407.

Package natbib Warning: Citation `ortiz2022atmosphere' on page 10
undefined on
input line 407.

Package natbib Warning: Citation `bonte2023towards' on page 10 undefined
on inp
ut line 407.

LaTeX Warning: `!h' float specifier changed to `!ht'.

Package natbib Warning: Citation `mebrek2020stream' on page 10 undefined
on inp
ut line 416.

Package natbib Warning: Citation `mebrek2020stream' on page 10 undefined
on inp
ut line 418.

Package natbib Warning: Citation `bolles2008streaming' on page 10
undefined on
input line 420.

Package natbib Warning: Citation `barbieri2010c' on page 10 undefined on
input
line 420.

Package natbib Warning: Citation `calbimonte2010enabling' on page 10
undefined
on input line 420.

Package natbib Warning: Citation `le2011native' on page 10 undefined on
input l
ine 420.

Package natbib Warning: Citation `anicic2011ep' on page 10 undefined on
input l
ine 420.

Package natbib Warning: Citation `komazec2012sparkwave' on page 10
undefined on
input line 420.

[10]

Package natbib Warning: Citation `le2012linked' on page 11 undefined on
input 1
line 422.

Package natbib Warning: Citation `le2013elastic' on page 11 undefined on
input
line 422.

Package natbib Warning: Citation `gillani2016dionysus' on page 11
undefined on
input line 422.

Package natbib Warning: Citation `le2013elastic' on page 11 undefined on
input
line 422.

Package natbib Warning: Citation `hoeksema2011high' on page 11 undefined
on inp
ut line 422.

Package natbib Warning: Citation `dia2018drss' on page 11 undefined on
input li
ne 424.

Package natbib Warning: Citation `dia2018drss' on page 11 undefined on
input li
ne 424.

Package natbib Warning: Citation `dia2018drss' on page 11 undefined on
input li
ne 426.

Package natbib Warning: Citation `khrouf2016waves' on page 11 undefined
on inpu
t line 428.

Package natbib Warning: Citation `khrouf2016waves' on page 11 undefined
on inpu
t line 428.

Package natbib Warning: Citation `khrouf2016waves' on page 11 undefined
on inpu

t line 431.

Package natbib Warning: Citation `calbimonte2016query' on page 11
undefined on
input line 433.

Package natbib Warning: Citation `calbimonte2016query' on page 11
undefined on
input line 433.

LaTeX Warning: File
`C:/ArticleLatexImagesFinal/AllinOneFrameWork_modified.draw
io.pdf' not found on input line 443.

! Package pdftex.def Error: File
`C:/ArticleLatexImagesFinal/AllinOneFrameWork_
modified.drawio.pdf' not found: using draft setting.

See the pdftex.def package documentation for explanation.
Type H <return> for immediate help.
...

l.443 ...al/AllinOneFrameWork_modified.drawio.pdf}

Try typing <return> to proceed.
If that doesn't work, type X <return> to quit.

[11]

Package natbib Warning: Citation `de2024enabling' on page 12 undefined on
input
line 460.

LaTeX Warning: File
`C:/ArticleLatexImagesFinal/Overall_Framework_Communication_
_OK.drawio.pdf' not found on input line 477.

! Package pdftex.def Error: File
`C:/ArticleLatexImagesFinal/Overall_Framework_
Communication_OK.drawio.pdf' not found: using draft setting.

See the pdftex.def package documentation for explanation.
Type H <return> for immediate help.
...

l.477 ...ll_Framework_Communication_OK.drawio.pdf}

Try typing <return> to proceed.

If that doesn't work, type X <return> to quit.

LaTeX Warning: File `C:/ArticleLatexImagesFinal/h3.drawio.pdf' not found
on input line 536.

! Package pdftex.def Error: File
`C:/ArticleLatexImagesFinal/h3.drawio.pdf' not
found: using draft setting.

See the pdftex.def package documentation for explanation.
Type H <return> for immediate help.
...

l.536 ...C:/ArticleLatexImagesFinal/h3.drawio.pdf}

Try typing <return> to proceed.
If that doesn't work, type X <return> to quit.

LaTeX Warning: File `C:/ArticleLatexImagesFinal/EDgeLayer.drawio.pdf' not
found
on input line 544.

! Package pdftex.def Error: File
`C:/ArticleLatexImagesFinal/EDgeLayer.drawio.p
df' not found: using draft setting.

See the pdftex.def package documentation for explanation.
Type H <return> for immediate help.
...

l.544 ...C:/ArticleLatexImagesFinal/EDgeLayer.drawio.pdf}

Try typing <return> to proceed.
If that doesn't work, type X <return> to quit.

[12]
Underfull \hbox (badness 5832) in paragraph at lines 559--560
[]\OT1/cmr/m/n/10 Queries de-fined in the DiSIF frame-work can
[]

Underfull \hbox (badness 2617) in paragraph at lines 559--560
\OT1/cmr/m/n/10 be cat-e-go-rized into in-de-pen-dent and de-pen-dent
[]

LaTeX Warning: File `C:/ArticleLatexImagesFinal/FogLayer.drawio.pdf' not
found
on input line 567.

! Package pdftex.def Error: File
`C:/ArticleLatexImagesFinal/FogLayer.drawio.pdf'
f' not found: using draft setting.

See the pdftex.def package documentation for explanation.
Type H <return> for immediate help.
...

1.567 ...icleLatexImagesFinal/FogLayer.drawio.pdf}

Try typing <return> to proceed.
If that doesn't work, type X <return> to quit.

[13] [14]
Underfull \vbox (badness 10000) has occurred while \output is active []

[15] [16] [17]

LaTeX Warning: File
`C:/ArticleLatexImagesFinal/Communication_new.drawio.pdf' n
ot found on input line 769.

! Package pdftex.def Error: File
`C:/ArticleLatexImagesFinal/Communication_new.
drawio.pdf' not found: using draft setting.

See the pdftex.def package documentation for explanation.
Type H <return> for immediate help.
...

1.769 ...ImagesFinal/Communication_new.drawio.pdf}

Try typing <return> to proceed.
If that doesn't work, type X <return> to quit.

LaTeX Warning: File
`C:/ArticleLatexImagesFinal/ConceptOnto_GraphViz_new.pdf' n
ot found on input line 779.

! Package pdftex.def Error: File
`C:/ArticleLatexImagesFinal/ConceptOnto_GraphV
iz_new.pdf' not found: using draft setting.

See the pdftex.def package documentation for explanation.
Type H <return> for immediate help.
...

1.779 ...ImagesFinal/ConceptOnto_GraphViz_new.pdf}

Try typing <return> to proceed.
If that doesn't work, type X <return> to quit.

Underfull \hbox (badness 1286) in paragraph at lines 796--804
[]\OT1/cmr/m/n/10 A query is sub-mit-ted through Fog-Worker1, which
[]

LaTeX Font Info: Trying to load font information for OMS+cmr on input
line 8
54.
(c:/texlive/2023/texmf-dist/tex/latex/base/omscmr.fd
File: omscmr.fd 2023/04/13 v2.5m Standard LaTeX font definitions
)
LaTeX Font Info: Font shape `OMS/cmr/m/n' in size <10> not available
(Font) Font shape `OMS/cmsy/m/n' tried instead on input line
854.
[18] [19] [20]

LaTeX Warning: File
`C:/ArticleLatexImagesFinal/result_TwoMaster_OneMaster.pdf'
not found on input line 928.

! Package pdftex.def Error: File
`C:/ArticleLatexImagesFinal/result_TwoMaster_O
neMaster.pdf' not found: using draft setting.

See the pdftex.def package documentation for explanation.
Type H <return> for immediate help.
...

1.928 ...agesFinal/result_TwoMaster_OneMaster.pdf}

Try typing <return> to proceed.
If that doesn't work, type X <return> to quit.

[21] [22]

LaTeX Warning: File `C:/ArticleLatexImagesFinal/cityOnto_graph.pdf' not
found o
n input line 972.

! Package pdftex.def Error: File
`C:/ArticleLatexImagesFinal/cityOnto_graph.pdf
' not found: using draft setting.

See the pdftex.def package documentation for explanation.
Type H <return> for immediate help.
...

1.972 ...ticleLatexImagesFinal/cityOnto_graph.pdf}

Try typing <return> to proceed.
If that doesn't work, type X <return> to quit.

Underfull \hbox (badness 1418) in paragraph at lines 1052--1053
[]\OT1/cmr/m/n/10 Continuing with the con-ges-tion de-tec-tion sce-nario,
[]

[23] [24]

LaTeX Warning: File
`C:/ArticleLatexImagesFinal/Time_execution_Q1Q2_StreamQR_No
nLogarithmicAdjustment_LinearScale_Q1Q2.pdf' not found on input line
1101.

! Package pdftex.def Error: File
`C:/ArticleLatexImagesFinal/Time_execution_Q1Q
2_StreamQR_NonLogarithmicAdjustment_LinearScale_Q1Q2.pdf' not found:
using draf
t setting.

See the pdftex.def package documentation for explanation.
Type H <return> for immediate help.
...

1.1101 ...arithmicAdjustment_LinearScale_Q1Q2.pdf}

Try typing <return> to proceed.
If that doesn't work, type X <return> to quit.

LaTeX Warning: File
`C:/ArticleLatexImagesFinal/Time_execution_Q1Q2_StreamQR_No
nLogarithmicAdjustment_LogScale_Q1Q2.pdf' not found on input line 1108.

! Package pdftex.def Error: File
`C:/ArticleLatexImagesFinal/Time_execution_Q1Q
2_StreamQR_NonLogarithmicAdjustment_LogScale_Q1Q2.pdf' not found: using
draft s
etting.

See the pdftex.def package documentation for explanation.
Type H <return> for immediate help.
...

1.1108 ...LogarithmicAdjustment_LogScale_Q1Q2.pdf}

Try typing <return> to proceed.
If that doesn't work, type X <return> to quit.

LaTeX Warning: File
`C:/ArticleLatexImagesFinal/Time_execution_Q1Q2_StreamQR_No
nLogarithmicAdjustment_LinearScale_Q3Q4.pdf' not found on input line
1116.

! Package pdftex.def Error: File
`C:/ArticleLatexImagesFinal/Time_execution_Q1Q
2_StreamQR_NonLogarithmicAdjustment_LinearScale_Q3Q4.pdf' not found:
using draf
t setting.

See the pdftex.def package documentation for explanation.
Type H <return> for immediate help.
...

l.1116 ...arithmicAdjustment_LinearScale_Q3Q4.pdf}

Try typing <return> to proceed.
If that doesn't work, type X <return> to quit.

LaTeX Warning: File
`C:/ArticleLatexImagesFinal/Time_execution_Q1Q2_StreamQR_No
nLogarithmicAdjustment_LogScale_Q3Q4.pdf' not found on input line 1123.

! Package pdftex.def Error: File
`C:/ArticleLatexImagesFinal/Time_execution_Q1Q
2_StreamQR_NonLogarithmicAdjustment_LogScale_Q3Q4.pdf' not found: using
draft s
etting.

See the pdftex.def package documentation for explanation.
Type H <return> for immediate help.
...

l.1123 ...LogarithmicAdjustment_LogScale_Q3Q4.pdf}

Try typing <return> to proceed.
If that doesn't work, type X <return> to quit.

[25] [26]
Underfull \hbox (badness 2119) in paragraph at lines 1181--1191
[]\OT1/cmr/m/n/10 Figures 11c[] and 11d[] re-veal an ex-po-nen-tial in-
[]

Overfull \hbox (4.24728pt too wide) in paragraph at lines 1251--1251
[] \OT1/cmtt/m/n/9 http://www.w3.org/2003/01/geo/wgs84_pos#location[]
[]

LaTeX Warning: File
`C:/ArticleLatexImagesFinal/Network_Usage_vs_Log_vs_Triples
.pdf' not found on input line 1278.

! Package pdftex.def Error: File
`C:/ArticleLatexImagesFinal/Network_Usage_vs_L
og_vs_Triples.pdf' not found: using draft setting.

See the pdftex.def package documentation for explanation.
Type H <return> for immediate help.

...

l.1278 ...nal/Network_Usage_vs_Log_vs_Triples.pdf}

Try typing <return> to proceed.
If that doesn't work, type X <return> to quit.

[27] [28]

LaTeX Warning: File
`C:/ArticleLatexImagesFinal/Memory_Consumption_vs_Stream_Ra
te_Linear_Enhanced_Q1Q2.pdf' not found on input line 1338.

! Package pdftex.def Error: File
`C:/ArticleLatexImagesFinal/Memory_Consumption
_vs_Stream_Rate_Linear_Enhanced_Q1Q2.pdf' not found: using draft setting.

See the pdftex.def package documentation for explanation.
Type H <return> for immediate help.

...

l.1338 ...vs_Stream_Rate_Linear_Enhanced_Q1Q2.pdf}

Try typing <return> to proceed.
If that doesn't work, type X <return> to quit.

LaTeX Warning: File
`C:/ArticleLatexImagesFinal/Memory_Consumption_vs_Stream_Ra
te_Log_Enhanced_Q1Q2.pdf' not found on input line 1345.

! Package pdftex.def Error: File
`C:/ArticleLatexImagesFinal/Memory_Consumption
_vs_Stream_Rate_Log_Enhanced_Q1Q2.pdf' not found: using draft setting.

See the pdftex.def package documentation for explanation.
Type H <return> for immediate help.

...

l.1345 ...on_vs_Stream_Rate_Log_Enhanced_Q1Q2.pdf}

Try typing <return> to proceed.
If that doesn't work, type X <return> to quit.

LaTeX Warning: File
'C:/ArticleLatexImagesFinal/Memory_Consumption_vs_Stream_Rate_Linear_Enhanced_Q3Q4.pdf' not found on input line 1353.

! Package pdftex.def Error: File
'C:/ArticleLatexImagesFinal/Memory_Consumption_vs_Stream_Rate_Linear_Enhanced_Q3Q4.pdf' not found: using draft setting.

See the pdftex.def package documentation for explanation.
Type H <return> for immediate help.

...

1.1353 ...vs_Stream_Rate_Linear_Enhanced_Q3Q4.pdf}

Try typing <return> to proceed.
If that doesn't work, type X <return> to quit.

LaTeX Warning: File
'C:/ArticleLatexImagesFinal/Memory_Consumption_vs_Stream_Rate_Log_Enhanced_Q3Q4.pdf' not found on input line 1360.

! Package pdftex.def Error: File
'C:/ArticleLatexImagesFinal/Memory_Consumption_vs_Stream_Rate_Log_Enhanced_Q3Q4.pdf' not found: using draft setting.

See the pdftex.def package documentation for explanation.
Type H <return> for immediate help.

...

1.1360 ...on_vs_Stream_Rate_Log_Enhanced_Q3Q4.pdf}

Try typing <return> to proceed.
If that doesn't work, type X <return> to quit.

Underfull \hbox (badness 1173) in paragraph at lines 1484--1485
[]\OT1/cmr/m/n/10 Resource Op-ti-miza-tion: Dis-tribut-ing com-pu-ta-tional

[]

No file DiSIF_Article.bbl.

[29] [30]

Overfull \hbox (5.24776pt too wide) in paragraph at lines 1572--1572
[]\OT1/cmtt/m/n/10 FROM STREAM <<https://www.wtlab.com/TrafficStream>>[]

[]

Overfull \hbox (5.24776pt too wide) in paragraph at lines 1600--1600
[] \OT1/cmtt/m/n/10 ?location concept:congestion "congestion" .[]
[]

Overfull \hbox (26.24757pt too wide) in paragraph at lines 1600--1600
[] \OT1/cmtt/m/n/10 FROM STREAM
<<https://www.wtlab.com/TrafficStream>>[]
[]

Overfull \hbox (5.24776pt too wide) in paragraph at lines 1634--1634
[] \OT1/cmtt/m/n/10 FROM STREAM <<https://www.wtlab.com/TrafficStream>>[]
[]

Overfull \hbox (5.24776pt too wide) in paragraph at lines 1679--1679
[] \OT1/cmtt/m/n/10 FROM STREAM <<https://www.wtlab.com/TrafficStream>>[]
[]

[31]

Package natbib Warning: There were undefined citations.

[32]

] (./DiSIF_Article.aux)

LaTeX2e <2023-11-01> patch level 1
L3 programming layer <2024-02-20>

LaTeX Warning: Label(s) may have changed. Rerun to get cross-references right.

)
Here is how much of TeX's memory you used:
11948 strings out of 474121
243195 string characters out of 5747949
1958190 words of memory out of 5000000
34057 multiletter control sequences out of 15000+600000
566609 words of font info for 69 fonts, out of 8000000 for 9000
1141 hyphenation exceptions out of 8191
94i,11n,93p,1829b,693s stack positions out of
10000i,1000n,20000p,200000b,200000s
<c:/texlive/2023/texmf-dist/fonts/typel/public/amsfonts/cm/cmbx10.pfb><c:/texlive/2023/texmf-dist/fonts/typel/public/amsfonts/cm/cmbx8.pfb><c:/texlive/2023/texmf-dist/fonts/typel/public/amsfonts/cm/cmcscl0.pfb><c:/texlive/2023/texmf-dist/fonts/typel/public/amsfonts/cm/cmexl0.pfb><c:/texlive/2023/texmf-dist/fonts/

typel/public/amsfonts/cm/cmml10.pfb><c:/texlive/2023/texmf-
dist/fonts/typel/pub
lic/amsfonts/cm/cmml5.pfb><c:/texlive/2023/texmf-
dist/fonts/typel/public/amsfon
ts/cm/cmml7.pfb><c:/texlive/2023/texmf-
dist/fonts/typel/public/amsfonts/cm/cmrl
0.pfb><c:/texlive/2023/texmf-
dist/fonts/typel/public/amsfonts/cm/cmrl2.pfb><c:/
texlive/2023/texmf-
dist/fonts/typel/public/amsfonts/cm/cmrl7.pfb><c:/texlive/202
3/texmf-
dist/fonts/typel/public/amsfonts/cm/cmrl8.pfb><c:/texlive/2023/texmf-dis
t/fonts/typel/public/amsfonts/cm/cmsy10.pfb><c:/texlive/2023/texmf-
dist/fonts/t
ypel/public/amsfonts/cm/cmsy6.pfb><c:/texlive/2023/texmf-
dist/fonts/typel/publi
c/amsfonts/cm/cmsy7.pfb><c:/texlive/2023/texmf-
dist/fonts/typel/public/amsfonts
/cm/cmtil10.pfb><c:/texlive/2023/texmf-
dist/fonts/typel/public/amsfonts/cm/cmtil8
.pfb><c:/texlive/2023/texmf-
dist/fonts/typel/public/amsfonts/cm/cmtil10.pfb><c:/
texlive/2023/texmf-
dist/fonts/typel/public/amsfonts/cm/cmtil8.pfb><c:/texlive/20
23/texmf-
dist/fonts/typel/public/amsfonts/cm/cmtil9.pfb><c:/texlive/2023/texmf-d
ist/fonts/typel/public/cm-super/sfrm1000.pfb><c:/texlive/2023/texmf-
dist/fonts/
typel/urw/zapfding/uzdr.pfb>

Output written on DiSIF_Article.pdf (32 pages, 372054 bytes).

PDF statistics:

217 PDF objects out of 1000 (max. 8388607)
139 compressed objects within 2 object streams
0 named destinations out of 1000 (max. 500000)
13 words of extra memory for PDF output out of 10000 (max. 10000000)

DiSIF: Distributed Semantic Information Fusion Framework for Smart City Applications

Ramin Rezvani-KhorashadiZadeh, Mohsen Kahani*

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

Abstract

In the past decade, the advancement of the Internet of Things (IoT) has facilitated the seamless implementation of integrated systems, generating continuous data streams. This data, characterized by high speed, volume, and heterogeneity, poses significant challenges. To address these challenges, an integrated model for heterogeneous data streams is essential for decision-making in many applications, including smart cities. The use of machine-readable semantic data models in stream reasoning and RDF stream processing is crucial for managing sensor data diversity. The JDL fusion model, particularly its semantic version, enables the integration of heterogeneous data using Semantic Web technologies. However, centralized fusion models are inefficient for the vast data volumes generated in smart cities, leading to network inefficiencies and privacy concerns. User queries, requiring data concept conversions, also suffer from increased execution times in centralized systems. The proposed Distributed Semantic Information Fusion Architecture (called DiSIF) addresses these issues using a three-layer architecture of edge, fog, and cloud, with worker and master nodes at each layer. This model performs low-level processing at the edge, sending only results to higher layers, thus enhancing network efficiency. Horizontal fusion integrates heterogeneous data to tackle sensor data diversity, while vertical fusion manages homogeneous data to reduce the transmission volume. The distributed approach ensures local data processing, while maintaining data privacy. Evaluations comparing the distributed and centralized JDL models depict that the DiSIF framework reduces network load, query execution time, and memory consumption, demonstrating its superiority for smart city applications.

Keywords: semantic JDL, data fusion, fog computing, stream processing, RSP agent.

1. Introduction

One crucial processing aspect in smart cities is the data fusion. With the advent of smart cities, multiple resources are made available for various applications. A common technique for managing multiple data resources is data fusion, which improves the quality of data outputs, aids decision-making, or extracts knowledge from raw data.

To meet the requirements of highly complex applications in smart cities, data from diverse sources must be utilized and fused to obtain valuable insights. Despite the existence of various data sources in different domains of smart cities, data fusion techniques that combine multiple data sources have become central to smart city frameworks. A single sensor alone cannot overcome physical limitations, such as a limited range in the surrounding environment. Therefore, combining information from various sensors provides a broader view of the surrounding environment (such as the surroundings of a vehicle), re-

ceived by sensors, enhancing the system's reliability regarding sensor errors.

On the other hand, these sensors have limited energy. Sometimes, some of these sensors may encounter problems, leading to errors in measurements. In summary, the reasons for the need for data fusion include system stability, improved situational awareness and inference, increased data accuracy, reduced uncertainty, expanded field of view, and cost reduction. Therefore, presenting a model for data fusion is one of the fundamental requirements in the field of smart cities.

In smart cities, data sources are heterogeneous, making the interpretation and combination of data challenging due to various interferences and diversities. Heterogeneity in data sources can be in terms of syntax, schema, or semantics. Syntax heterogeneity arises from differences in language, schema heterogeneity from differences in structure, and semantic heterogeneity from differences in the meaning of data [4], [5]. The absence of a common understanding in the fusion system leads to the loss of system coherence and the emergence of semantic conflicts during data fusion. Therefore, one of the main challenges in fusion systems is semantic challenges and the lack of a common understanding across these systems. To address this, it is

*Corresponding author

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

necessary to focus on conceptualizing the target domain and provide an explicit and formal expression of concepts and features to establish a common description of the domain.

Various models have been proposed for data fusion [6]. The JDL fusion model is a successful and prevalent model in the field of data fusion, with the semantic version of JDL creating the capability to integrate heterogeneous data using Semantic Web technology [7]. The semantic JDL model [7] allows different architectures to be used for implementation. These architectures are classified as centralized, distributed, and hybrid.

Existing models for semantic JDL model have been centralized, where data is placed in a single node, and fusion operations are performed at that node. The centralized semantic JDL model can lead to the following problems. First, as given the vast volume of data generated in smart cities, a centralized approach to data fusion models cannot be efficient. In smart cities, given the massive volume of data and various decision-making levels, decision-making should be carried out at different levels for better management. Many important and comprehensive decisions in smart cities require information from the environment that is not easily accessible. To obtain such information, processing, fusion, and decision-making on lower-level data are necessary.

Therefore, for smart city applications, multi-layered decision-making is necessary. In the centralized JDL model (traditional JDL model) and other fusion models, there is only one level and one perspective for decision-making, and data are fused only at one level. Therefore, it is necessary to collect all data in one node and one level, and this large volume of data transmission between nodes reduces the network efficiency. Therefore, making decisions on a large and comprehensive scale can be time-consuming and costly. Hence, centralized fusion model is not suitable for smart cities.

Second, decisions made in smart cities based on data can be divided into two main groups. The first group includes time-sensitive decisions that need to be made with minimal delay and in the fastest possible time with minimal data processing. The second category includes decisions that require extensive processing on a massive volume of data and are less sensitive to delays compared to the first category.

Therefore, the proposed framework should be able to response to these two types of decisions. Therefore considering various frameworks proposed for smart cities [8], a framework based on the three-layer architecture of edge, fog, and cloud is chosen for this article. In this framework, the edge layer is used for performing first category decision-making processes that are time-sensitive. The fog and cloud layers are utilized for second-category decisions that require heavier processing. Therefore, to use the distributed semantic JDL model in smart cities, this article designs a framework based on the three-layer architecture of edge, fog, and cloud for the distributed semantic JDL

model. One of the issues present in the centralized semantic JDL approach is the increasing response time to user queries. User queries can be analyzed from two perspectives in general. The first aspect concerns the volume of data required to respond to user queries. Queries that include a large area may need to be broken down into sub-queries for smaller areas and then aggregated. Therefore, executing these queries in a centralized manner can be time-consuming.

On the other hand as the second aspect, users may submit queries for high-level concepts (such as querying for traffic in an area). Executing these queries requires data that is not captured by sensors and needs to be generated through a series of processing steps. Thus, executing these sub-queries and then the main query can take a considerable amount of time in the centralized semantic JDL approach.

To address the above issues, fusion in the DiSIF framework is performed in two ways: horizontal fusion and vertical fusion. Vertical fusion is a fusion that leads to a change in the scale of data, where the concept of the data does not change, but their scale increases. For example, combining traffic data from several streets and generating traffic flow for an entire area. In this fusion, the perspective changes from a fragmented view to a comprehensive one, and the decisions made are for a larger scale. In this fusion, the user query is broken down into several independent sub-queries through the cityOnto ontology. Each of these sub-queries is executed in parallel in the master nodes of the lower layer, and then the results are sent to the worker nodes in the upper layer, where the fusion takes place. This way, the execution time of the query and the resources consumed for its execution decrease compared to the centralized approach.

Horizontal fusion leads to a change in the concept of the data, where data streams with different and heterogeneous concepts, such as vehicle movements, congestion concept and etc, are received from different nodes and fused to produce a new concept (e.g., traffic concept). The user query is divided into dependent sub-queries. This concept generation is done through the conceptOnto ontology, and each of the dependent sub-queries is performed in worker nodes for the corresponding concept, while the main query is carried out in the master node of the same layer. In this way, the execution time of these types of queries also significantly decreases compared to the centralized approach. Therefore, horizontal fusion is a fusion that results in the generation of a new concept through the fusion of data streams from different domains.

The DiSIF framework has additional features as follows:

- **Execution of Independent Queries in Parallel**

An independent query can be broken down into sub-queries, and each sub-query can be executed independently in different nodes in a parallel and efficient manner. The results obtained from the execution are

then sent to the central node and combined. Experimental results show that using a distributed approach for parallel execution of sub-queries can significantly optimize query execution time. The combination of these sub-queries is performed in worker nodes of each layer, and the required sub-queries are executed in master nodes of the lower layer. Therefore, the DiSIF should have the capability to break down queries into independent sub-queries, execute each one separately, and finally combine and generate the final output.

- **Execution of Complex and dependent Queries**

These complex queries are not predefined in the system and are introduced during runtime. They require other queries to have been previously executed in the system. The execution of these queries is performed in master nodes of each layer, while the required pre-queries are executed in worker nodes of the same layer. Recent work has focused on executing predefined queries or tasks, which differs from the dynamic execution of complex queries. The DiSIF efficiently supports the dynamic execution of complex queries

- **Reduction of Network Load**

Instead of sending raw and unprocessed data over the network (as in a centralized approach), processed data are sent and received. The volume of processed data is much smaller than raw data, significantly reducing network bandwidth utilization.

- **Enhancement of Data Privacy**

By avoiding the transmission of raw data over the network and adhering to data privacy in the distributed approach, data privacy is maintained. In contrast, in the centralized approach, raw data must be collected in a central node, compromising data privacy.

As stated earlier, the primary goal of this article is to present a distributed version for semantic JDL fusion. While various advancements have been made in different aspects of JDL fusion models, a distributed approach for JDL has not been explored until now. The innovations proposed in this article can be outlined as follows:

1. **Distributed Semantic JDL Fusion Model Across Three Layers (Edge, Fog, and Cloud)**

This section elaborates on the decision-making at different levels and in a multi-layered fashion, combining macro decisions by fusing micro and small decisions. The DiSIF framework allows high-speed execution of macro decision-making operations parallelly across different levels, outperforming the centralized JDL fusion approach. To enhance speed, the separation of fusions and decision-making has been considered. Many macro decisions require preparation and utilization of lower-level decision-making (smaller fusions), and in the centralized JDL model, all fusion

operations and outputs are placed on a common BUS platform without separation. This article addresses the speed and complexity issues by separating and isolating fusions, making decisions at different layers, and performing parallel fusions in different layers.

2. **Proposing two fusion models: horizontal fusion (data concept change) and vertical fusion (data scale change)**

Previous fusion models operated on data from a specific concept. The DiSIF framework can fuse data from various concepts, such as traffic and weather, generating new concepts or natures. Additionally, in vertical fusion, data from a similar concept but on a smaller scale are fused, forming a larger data stream. This innovation allows for macro and precise decision-making with a broader perspective on the surrounding environment.

3. **Introducing a distributed query execution model**

This article introduces an optimized approach for executing queries in a three-layer model in a distributed manner. Independent queries are executed between layers and by worker nodes in the upper layer and master nodes in the lower layer. Dependent queries are executed within layers and between worker and master nodes in each layer. This optimized approach not only reduces query execution time but also minimizes the required memory for query execution and optimizes network bandwidth usage.

It is worth noting that the article does not present innovations in the cloud domain. The advantages of the cloud are utilized for complex fusions and macro-level decision-making in the city. Decisions from the fog layer are sent to the cloud node for macro-level decision-making.

The structure of the article is as follows: In section 2, related works on data fusion models, Semantic Web approaches in smart cities, fog computing architectures, and RDF stream processing approaches are discussed. In section 3, the DiSIF framework is described from the perspectives of the JDL model and the fusion model. Each layer is then discussed separately. Evaluations are conducted from various perspectives and the results are analyzed in section 4 and the conclusion is presented in section 5. Additionally, the queries used in this article are provided in the Appendix Appendix A .

2. Related works

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

2.1. Data fusion methods

The process of data and information fusion involves combining inputs to create richer information than what can be obtained from each input separately. Techniques for data and information fusion were first introduced in the 1970s. The applications of these techniques are vast, initially used primarily in military contexts but now expanded to non-military domains as well.

Generally, data and information fusion can improve the output of processes for solving many different applications. Some of the benefits of using data and information fusion in problem-solving include Improving the accuracy and precision of data, such as reducing uncertainty and ambiguity, and enhancing awareness of situations and reasoning, leading to better decision-making.

The extensive applications of fusion in various domains (such as sensor networks, weather forecasting, robotic exploration, military command and control systems, crisis management, etc.) have led to the development of numerous models and architectures for data fusion. These models are broadly categorized into three main types: information-centric models, activity-centric models, and role-centric models.

Information-centric models focus on the data generated during the fusion process. Common information-centric models include the well-known JDL model and the DFD model. Activity-centric models focus on the steps and activities required for the fusion process. Examples of activity-centric models include the Boyd control cycle model and the Omnibus model. Role-centric models consider how fusion is designed and modeled, taking into account the necessary roles and modeling their interactions. The Frankel-Bedworth model is an example of a role-centric model.

One of the common models in the field of fusion is the multi-level JDL model [?], which with its 5 different levels covering from raw data processing to decision-making, has contributed to the comprehensiveness and popularity of this model in the fusion domain. For this reason, extensive research has been conducted based on the JDL over the years. This model was first introduced in 1987 by the JDL group at the US Department of Defense, defining various levels which are outlined below:

Level 0 (Preprocessing): At this level, preprocessing operations are performed on various data sources, and cleaned data is returned as the output.

Level 1 (Object Refinement): In level 1, further analysis pertains to individual objects or targets. Essentially, identity, parametric, and positional information is summarized and presented for each object separately. Therefore, the output of this level includes objects and their attributes.

Level 2 (Situation Refinement): At this level, based on prior knowledge and environmental information, a conceptual description of the relationships between objects is created. Since entities along with their attributes (which were outputs of level 1) are present at this level, a general visual representation of the current situation is determined

by establishing object positions and relationships between them.

Level 3 (Threat Refinement): The tasks of this level include evaluating the current situation and predicting threats and vulnerabilities. Additionally, deriving solutions and action opportunities in the emerging situation is the responsibility of this level. It can be said that the products of processing layers 1 to 3 of the JDL model consist of: Layer 1, object representations; Layer 2, representations of inter-object relationships; Layer 3, representations of the effects of inter-object relationships.

Level 4 (Process Refinement): At level 4, monitoring the system's performance and allocating resources based on defined objectives takes place. In other words, measuring system performance and its effectiveness fall under the responsibility of this level. Continuity of monitoring, selection of managerial information, and service quality creation are performed at this level. Essentially, process refinement ensures that data fusion is carried out in such a way that optimal results are obtained for the system. This stage can improve the final system results by adjusting parameters in the data fusion process, creating prior knowledge of objectives, or directing sensors to achieve better coverage of the search area.

The database management system utilizes the JDL model of a database management system that performs data update operations and prepares information for the fusion process. Given the large volume of data in a fusion process, the existence of such a system seems necessary. Another part of this model is human-computer interaction, which serves as a vital interface for human input and communicates fusion results to users.

The semantic JDL model [?] combines the basic JDL model with Semantic Web technologies. The overall structure of the JDL semantic model is illustrated in Figure 1.

Level zero (Pre-processing): At this level, preprocessing operations are performed on various data sources, and the cleaned data is provided to the object refinement level. Therefore, semantics are not introduced at this level.

Level one (Object refinement): This level is also not fully semantic. At this level, we only transform objects and their attributes into standard RDF format and store them. Other tasks at this level, such as identifying objects and relating attributes to objects, are not semantically defined and are performed using mathematical algorithms and image processing concepts. Essentially, we represent existing data semantically at this level for use at higher levels.

Hence, using pre-defined ontologies, objects along with their discovered attributes are imported into RDF format, and with the help of RDFizer, object-related data along with their attributes are converted to RDF format and stored in the RDFStore database. Therefore, it is evident that for all relevant domain concepts, an ontology needs to be defined so that at this level, objects along with their related attributes can be structured.

Level two (situation refinement): At this level, based

Table 1: Comparison of Activity Separation in Data Fusion Models

Model	Data Collection	Signal Processing	Object Refinement	Location Refinement	Threat Refinement	Decision Making	Acting
JDL Model	—	Level0	Level1	Level2	Level3	Level4	—
OODA Loop	Observe	Observe	Orient	Orient	Orient	Decide	Act
Intelligence Cycle Model	Collection	Collection	Collection	Evaluation	Evaluation	Dissemination	Dissemination
Omnibus Model	Sensing and Signal Processing	Sensing and Signal Processing	Feature Extraction	Decision Stage	Decision Stage	Decision Stage	Act Stage
Object-Oriented Model	Actor	Actor	Perceiver	Perceiver	Perceiver	Director	Manager

Table 2: Comparison of Data Fusion Models based on Data, Model, and Data Fusion Type

Data	Model	Data Fusion Type	Differentiation between Situation and Threat Refinement
JDL Model	Data-Based	Shared Memory (BUS)	✓
OODA Loop	Activity-Based	Sequential	✗
Intelligence Cycle Model	Activity-Based	Sequential	✗
Omnibus Model	Activity-Based	Sequential	✗
Object-Oriented Model	Role-Based	Sequential	✗

on prior knowledge, environmental information, and data collected at level one (information related to entities along with their attributes in RDF format), the situation of objects and explicit relationships between them are determined (previously possible relationships between objects have been defined).

Then, further tasks at this level, based on discovered relationships, identify some attributes related to objects that were previously unknown, and new relationships are extracted from existing relationships and attributes using inference. Therefore, it is evident that at this level, new information is obtained or previous information changes, so new inferential information about the situation is imported into RDF format by RDFizer and stored in RDFStore.

Level three (Threat refinement): Evaluating the current situation and predicting threats and vulnerabilities are tasks at this level. Therefore, a semantic reasoner is essential at this level. At this level, using semantic inference techniques and the embedded reasoner, and considering the rules existing in the rules base, the current state assessment and complete inference for predicting threats are performed.

Then, considering that finding solutions and opportunities for identified threats is the responsibility of this level, a list of opportunities will also be available as output. At this level, ontologies for threats and opportunities are defined. The outputs of this level are sent to RDFizer to be stored in RDF format after conversion.

Level four (Process refinement): At level four, monitoring system performance and resource allocation based on defined goals take place. At this level, an expert, by observing the outputs of level three, makes necessary decisions and makes general changes to the system to improve efficiency.

Database management in JDL semantic involves multiple databases. One database is considered for storing information in RDF format, including the RDF schema and storing ontologies. For the rules existing in the domain, a rules database will also be available.


For all these databases, principles of database management should be considered, and necessary aspects of database management such as compatibility, updates, insertion, and deletion, and editing should be performed at an acceptable speed. Some other details of the model and precise inputs and outputs are definable depending on the domain for which the model is implemented.

In Table 1 a comparison of the activity breakdown in data fusion models is presented.

In Table 1, each activity is specified to indicate where it fits within a data fusion model. The most notable observation from Table 1 is that, except for the JDL model, all other models cannot fully separate tasks and activities. For instance, in the Boyd control loop model, both data collection and signal processing activities are performed within the observation component, or in the omnibus model, object refinement and situation operations are carried out within the feature extraction component. Therefore, one of the notable features of JDL, which makes it suitable for distributed architecture, is its ability to effectively separate tasks.

Additionally, in Table 2, a comparison between data fusion models is provided. In Table 2, one of the comparison criteria is the differentiation between situation refinement and threat refinement in the fusion model. In the domain of smart cities, we need separate and distinct discussions for traffic discovery (situation refinement) and traffic prediction (threat refinement) for the future. Therefore, a model that can separate these two tasks is more suitable for use in smart cities.

As observed in Table 2, one of the advantages of the JDL fusion model over other fusion models is the separation of fusion stages into completely distinct parts. This allows for the distribution of different components of the JDL model across various systems, enabling distributed fusion operations. On the other hand, in existing fusion models (excluding JDL), the two stages of situation refinement for discovering the current system position and threat refinement for predicting the future system status



C:/ArticleLatexImagesFinal/Semantic_JDL.jpg

Figure 1: The semantic JDL model [?]

are presented inseparably. This approach is not suitable for the smart city context, where situation detection and future prediction are entirely different tasks. Therefore, in this article, the JDL fusion model is utilized to present its distributed version for smart traffic application.

2.2. Semantic Web in smart city

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

For achieving data interaction through Semantic Web, in [?], a layered architecture called LSM is introduced which integrates sensor data using Semantic Web. This architecture defines a user interface for publishing, annotating, and querying sensor data. It also utilizes the SSN ontology to describe sensor data and sensor data streams. In this approach, time-dependent data is linked with linked data resources. This linkage is established by adding semantic annotations to sensor resources and sensor data streams.

The LSM architecture comprises different layers. In the data acquisition layer, various converters for covering a wide range of input data formats are present, and at the output of this layer, data are sent to other layers in a uniform and standardized triple format. The presence of converters enables a form of interaction in the LSM architecture. In the linked data layer, semantic information from

linked data sources such as DBpedia, GeoNames, etc., is added to the input data streams. In the data access layer, semantic query engines for performing various queries on linked streaming data are placed, which can be pull-based or push-based. In the application layer, various applications are placed as user interfaces in this section. This approach does not address query decomposition and distributed query execution and does not consider optimizing query execution. Additionally, it considers the execution of CQELS queries centrally and does not consider the cost of sending data to the central node.

In [?], a framework called SDFE is proposed for integrating various data from different sensors. This article utilizes Semantic Web technology to address the problem of inconsistency between received data. It focuses on discovering semantic conflicts between heterogeneous sensor data. These conflicts can arise in the data measurement units. Data measurement units play an important role in the data fusion process, data interpretation, and comparisons [?].

In this framework, the raw data collection layer is responsible for collecting data from connected sensor networks. The data storage layer is intended for storing data. In this layer, two storage sources are considered, one for raw sensor data (non-semantic) and one for stored semantic data. The semantic annotation and mapping layer are responsible for adding semantic annotations to sensor descriptions and observations. The data dissemination layer is responsible for publishing aggregated data through a standard user interface and providing data to different sections. In the data processing layer, the discovery and res-

olution of measurement unit conflicts that may arise between heterogeneous sensor data are performed. The data visualization layer comprises various tools for displaying outputs obtained in this layer. This article does not discuss topics such as distributed processing and query execution, query execution optimization, processing of large streaming data, and prevention of sending data to only one central node (centralized approach).

In [?], a framework is presented for combining and aggregating heterogeneous data streams from different sensors with the aim of transforming data streams into feature streams. Given the large volume of data being generated, combining them during production can significantly reduce data streams and increase efficiency.

In [?] Sensor data streams are transformed into RDF based on domain ontologies and then into features, and feature streams are published instead of data. This article focuses on reducing the volume of transmitted data in the network and optimizing bandwidth consumption. However, it does not discuss query processing, query execution optimization, and three-layer architecture.

In [?], for better communication and aggregation between data, information or sensor features are also published. Sense2web [?] is a framework in which the main features of sensors are extracted and represented semantically (RDF) to establish necessary links to other resources [?]. Therefore, in addition to communications between data streams, communications between sensors are also considered, and sensor descriptions are manually published.

In [?], a new architecture for aggregating heterogeneous sensor networks, converting measurements into semantic data, and performing inference processes on them is introduced. This architecture focuses on data received from sensors, combining different ontologies, and semantic inference. In this article, data is received from sensors and annotated semantically. Then, in the semantic-based application section, they are combined and aggregated. This article also does not discuss the presentation of a distributed three-layer architecture for data collection, distributed query processing, bandwidth management, and processing of large data streams.

Another architecture in [?] is introduced for integrating heterogeneous information from multiple sources in IoT. In this architecture, heterogeneous data from multiple sources are initially integrated to structure IoT data and improve data quality. In the next step, information fusion is performed on these structured data.

In this article, a data aggregation layer is designed to aggregate heterogeneous data from various sources. This layer focuses on aggregating related data from heterogeneous sources that may differ structurally, content-wise, or in value and may have overlaps. This data aggregation creates a unified view of these data for users to enhance data quality. The data fusion layer emphasizes on aggregating and merging structured data to make appropriate decisions. This architecture, by collecting all data from vari-

ous sensors in a central node, performs data fusion through a data integration layer to improve data fusion. Considering a three-layer and distributed architecture, data can be fused in lower layers and in corresponding nodes to optimize system performance. Therefore, adopting a centralized approach for data processing, the need to send all data to a central node, and the lack of providing a model for optimizing query processing are some drawbacks of this approach.

Another introduced architecture for collecting and disseminating sensor data from various sources is the SIGHTED architecture [?]. One of the issues with the SIGHTED architecture is that it places annotated data in a storage space and then performs queries on this stored data. This process increases the time required to extract annotated data for each query. Hence, the challenges of using a centralized approach for data collection, query processing, lack of query optimization, inability to handle large data streams, and the need to send all data to a central node are some drawbacks of this approach.

One of the proposed semantic frameworks for sensor data integration is presented in [?]. In this framework, data is collected from sensors, high-level operations are performed on them, and a unified view of the sensor network is presented to the user. This architecture consists of three layers: the data layer for discovering, collecting, and aggregating data; the processing layer for processing the collected data; and the semantic layer for ensuring semantic compatibility and content annotation using ontologies. The semantic layer includes sections such as: Rules, where two distinct sets of rules (mapping rules and semantic rules) are present; Ontology model; Inference server capable of inferring facts from existing facts in the ontology; Semantic queries providing access to knowledge base information. The architecture presented in this article is in a pipeline form and is not distributed, so data processing is not distributed and is inefficient. Also, the data ultimately resides in a central node to respond to user queries, so query execution is also centralized. Issues such as optimizing query execution and managing the transmission of large data are also not addressed.

In [?], it employs Streaming Virtual plays Streaming ViKnowledge Graphs for processing and merging semantic data streams, utilizing RDBMSs and the ontology-based data access (OBDA) approach as an ontology constructed from several heterogeneous relational data sources. This article introduces different version of OnTop ontology for processing big data stream. Therefore, OBDA ontologies are generated based on RDBMS relational databases and used in merging streaming data. The process of constructing an ontology from RDBMS databases in this article can be time-consuming. In smart cities where data is often decentralized, this approach may not be bandwidth-efficient. Additionally, the capability for query decomposition and efficient query execution is not provided in this approach.

In [?]s, the integration of IoT solutions in smart cities generates vast amounts of data that need to be exchanged

via APIs, yet achieving true semantic interoperability remains a challenge due to the heterogeneity of IoT platforms. The study highlights the use of ontologies with formal semantics and shared vocabularies to describe environmental sensing and wellness monitoring, addressing these interoperability issues. By leveraging sensor-agnostic APIs and MCS-dedicated ontology modules, smart cities can enhance data integration, support complex functionalities, and improve scalability and real-time responsiveness in IoT applications.

Privacy is a significant concern in Smart Cities due to the vast amount of data generated and shared by IoT devices. Ensuring that personal and sensitive information is adequately protected while maintaining high service quality is a critical challenge. [?] proposes the 'Ontology-Based Privacy-Preserving' (OBPP) framework, which addresses critical challenges in Smart Cities such as heterogeneity, privacy preservation, and high-level service provision through a three-module approach. The framework's use of ontologies and semantic reasoning demonstrates its effectiveness in improving service quality and maintaining robust privacy measures in smart city environments.

Semantic Web technologies in addition to their applications in smart cities, they play a crucial role in Agriculture 5.0 by enabling sophisticated spatio-temporal and semantic data management systems that enhance data interoperability, accessibility, and real-time operations in the agricultural sector [?].

2.3. Fog Computing architectures

Recently, cloud computations with nearly unlimited resources have emerged as a promising paradigm to tackle the high complexity of computational tasks associated with smart cities [?]. However, its inherent constraints, such as significant delays, unawareness of contextual behavior, and lack of support for mobility, pose serious limitations in its utilization in real-time smart environments. Apart from these incidental risks, cloud computations suffer from insufficient processing speed due to the voluminous data generated by smart city devices.

On the other hand, edge computing extends the capabilities of cloud computing to the network edge, offering features such as awareness, low latency, mobility support, and scalability. Therefore, edge computing presents a suitable solution for overcoming the limitations of cloud computations in real-time smart urban environments [?] - [?].

Edge computing, by providing processing and storage resources with minimal delay, offers a promising approach to enable smart city applications. It transfers computational resources such as computing power, data, and applications from remote clouds to the network edge. Consequently, it enables many smart city services to be realized in real-time. Recently, three different technologies, namely cloud computing, fog computing, and edge computing, have been widely discussed in the literature to bring notable features of cloud computations closer to the

edge [?]. For instance, edge computing reduces data transmission from end-users to remote clouds, thereby reducing network bandwidth usage [?]. Numerous articles have been proposed for the application of fog computing in smart cities.

Perera [?] provides an overview of fog computing technology. Various real-world scenarios including smart agriculture, intelligent transportation, smart healthcare, smart waste management, smart water management, smart greenhouse gas control, smart power grids, and smart retail automation have been discussed. However, this article does not address Semantic Web-based data fusion approaches.

In [?], the architecture, technologies, and applications of fog computing are examined. Different aspects of fog computing, edge computing, and cloud computing are compared. The applications of fog computing in various case scenarios, including smart healthcare, smart environments, and transportation, are discussed. Additionally, a hierarchical architecture of fog computing is presented. Finally, challenges and open issues in this area such as security and privacy concerns, energy consumption, and service management are provided. This article does not discuss Semantic Web approaches.

Shi [?] describes edge computing and its advantages and disadvantages. Several case studies including smart homes, video analysis, cloud offloading, and smart cities are presented. Various challenges such as naming, data abstraction, privacy and security, optimization metrics, and service management are discussed that can be addressed by edge computing. While sometimes fog and edge computing are used interchangeably, this article primarily focuses on edge computing over fog computing.

Recent research trends in fog computing and the Internet of Everything (IoE) are depicted in [?]. Major technology challenges including latency reduction, rational bandwidth usage, IoE device resource constraints are discussed, and fog computing can provide solutions. Scalability, security, programming, and real-time features are considered as the main features of fog computing. In this article, cloud computing and fog computing are compared with different perspectives. Furthermore, four use cases including IoE and smart network, smart cities, big data streams, and industry are provided to address open issues in fog computing. Smart cities are only introduced as a use case, and detailed aspects of fog computing applications in smart cities are neglected. The application of fog computing in smart cities is a shadowed topic in this article.

In [?], the primary idea is to utilize intelligent agents in smart city applications. A three-layer architecture called Rainbow is proposed to advance the proposed approach. Three real-world case studies related to Cosenza, Italy are discussed to describe the proposed approach. They claim that the use of intelligent agents enables IoT systems to support real-time applications in smart cities. This article does not discuss the applications of Semantic Web tech-

nology in data fusion and how to respond to user queries.

In [?], a tiered-edge architecture is introduced, providing the capability to distribute requests across nodes. This architecture consists of three parts: the request dispatching component responsible for sending requests to the system, the processing distribution component performing distributed request processing, and the result delivery component responsible for sending back the results of request execution. This architecture introduces a new semantic stream processing architecture that breaks down each request into sub-requests, allowing each sub-request to be executed on independent nodes. In this approach, a mechanism is created to control and balance the workload of requests. In this article, the user query is divided into several independent queries based on the streams required for execution, and each query is executed on an independent node, and then the results are returned.

Therefore, if a query is written on high-level concepts, there is no capability to break down the query into lower-level concepts (by ontology) for execution on nodes. In other words, there is no possibility of breaking down the semantic meaning of the user query. On the other hand, the possibility of breaking down a stream into several sub-streams does not exist in this system. Additionally, the system allows for the execution of duplicate sub-queries. The following frameworks in smart cities are described in detail.

FogBus [?] provides a framework for developing and deploying integrated cloud-fog environments with structured communications for data processing. In this framework, two types of nodes are used: worker nodes for performing processing tasks and work assignment requests, and broker nodes for assigning tasks to different worker nodes. In this framework, various Internet of Things (IoT) sensors send data and tasks to fog nodes with the help of gateway devices. Resource management and task preparation are performed in fog broker nodes. When the fog infrastructure becomes overloaded, FogBus activates resources from cloud data centers. This strengthens the system, enables fast execution of heavy tasks, and also decentralizes data processing. However, this architecture does not discuss semantic data processing and query decomposition approaches.

The article [?] presents a new architecture called "Analytics Everywhere" based on edge-fog-cloud for analyzing Internet of Things (IoT) streaming data in smart parking scenarios. In this architecture, edge, fog, and cloud nodes are hierarchically utilized. After receiving streaming data, it is sent to edge nodes for processing and preprocessing, then forwarded to fog nodes for more complex processing, and finally to cloud nodes.

In [?], the edge nodes receive streaming data from sensors and send it to a message broker, which then forwards the data to fog nodes, pre-designated for managing these edge nodes. The data then moves from fog to cloud nodes. Each layer performs processing. Edge nodes handle processing for descriptive analytics (statistical output)

for performance analysis and real-time event examination. Fog nodes handle event detection analysis, investigating the causes of events in the stream. Cloud nodes perform predictive analysis, forecasting future streams in the architecture. The main objective of the Analytics Everywhere architecture is automating predefined analytical tasks for edge, fog, and cloud nodes for these analyses. This architecture does not discuss optimal execution of user requests, preventing duplicate execution of user queries, breaking down queries into different sub-queries for efficient execution, and the use of RDF semantic technology for data and their fusion.

Aryaian et al. [?] proposed a four-layer architecture for fog, including data generation, cloud computation, fog computation, and data consumption layers. Consumers can send their requests to the other three layers and receive required services. The data generation layer hosts IoT devices, communicating with the fog computation layer and the cloud computation layer. In this architecture, data preprocessing is performed in the fog computation layer. This layer also considers context awareness and low latency. The cloud computation layer provides centralized control and a wide range of monitoring services. Long-term and complex behavior analysis is performed in this layer to support long-term pattern recognition and large-scale event detection. The main difference of this architecture from others is the direct interaction between consumers in all three layers.

In the fog processing layer, many low-power computational nodes are located. Each node in this layer is responsible for processing data from neighboring sensors forming a community. The output of the fog layer has two parts: the first part sends processed data for further analysis to the upper layer, and the second part sends results to users. The cloud layer is responsible for data analysis at a larger scale and scope. Complex and long-term analyses on a wide scale can be performed in this layer, such as large-scale event detection, long-term pattern recognition, etc. In this architecture, outputs from each layer can be directly sent to users. One issue in this architecture is the lack of adoption of Semantic Web and the absence of providing a data fusion model from sensors since diverse data in a smart city context need a model for their fusion and effective utilization. Additionally, optimal execution of user requests is not discussed.

Dastjerdi and colleagues [?] proposed a five-layer fog computing architecture. The topmost layer is the Internet of Things (IoT) application layer, which provides application functionalities to end-users. The next lower layer is the resource management defined by software, which deals with issues related to monitoring, provisioning, security, and management. The subsequent layer is responsible for managing cloud services and resources. The next lower layer is the network layer, which works to maintain communication between all devices and the cloud. The bottommost layer consists of end devices such as sensors and gateways. In this architecture, there is no fully designated

fog layer, and it also does not indicate where the computation is performed. Additionally, it does not address semantic issues or how data fusion is performed.

In [?], a collaborative architecture for IoT is used, which consists of edge, fog, and cloud nodes. These nodes interact collaboratively through agent-oriented algorithms via software agents and Complex Event Processing (CEP). The data flowing through this system are not of semantic type, and there is no provision for heterogeneous data utilization in this architecture. Moreover, the possibility of breaking down queries and optimizing user requests in the form of queries is not considered.

In [?], it is stated that due to the high volume of data generated in the IoT domain, it is not efficient to upload all data to the cloud as it causes network congestion, low throughput, high latency, and privacy issues. To solve this problem, edge processing is proposed. In this article, a general approach for edge processing is presented, which can identify which components should be offloaded to the edge layer. In other words, a generic CR edge processing platform is proposed in this article, which can identify which parts of queries should be offloaded to the edge layer and execute queries efficiently. In this article, privacy, low-latency, and autonomous processing directly at the edge are addressed. Moreover, since fewer data need to be sent to the cloud, higher throughput in the cloud and less network congestion are also provided. On the other hand, in this approach, the automatic translation of high-level queries into specific queries and their automatic offloading to the edge layer to enable generic edge processing does not take place. Additionally, the multi-node and master-slave approaches are not used in conjunction with the capabilities of the cloud, fog, and edge layers in this article. Therefore, load balancing is not provided in this approach.

Edge computing layers, driven by IoT advancements, enhance semantic data enrichment by pre-processing and structuring data close to the source. This approach, highlighted in a recent study, improves data quality and enables efficient reasoning and complex event processing in smart city applications [?].

In fog computing architectures, ensuring data integrity is challenging due to the distributed and dynamic nature of the environment. Traditional centralized methods for verifying data integrity become inadequate in this context, leading to high latency and potential bottlenecks. To address these issues, [?] introduces an efficient public verification protocol that leverages the short integer solution problem (SIS) and identity-based signatures. This protocol allows any legitimate end user to verify data integrity without relying on centralized third parties, significantly improving security and efficiency.

2.4. RDF stream Processing approaches

Processing a vast volume of data streams in real-time has emerged as a significant concern in recent researches. To address the challenge of heterogeneous data, the RDF

model is proposed, alongside systems for RDF stream processing (RSP) and languages for continuous querying.

Numerous RSP systems are documented in the literature, with some employing complex temporal operators such as EP-SPARQL [?], SPARKWAVE [?], and INSTANS [?]. Conversely, other systems utilize sliding windows, exemplified by C-SPARQL [?] and CQELS [?].

In general, various approaches for implementing RSP systems can be categorized into the following types:

1. Distributed models that are not open-source, such as CQELS Cloud [?], or those designed on top of frameworks like Spark Streaming, Flink, or Apache Storm. For instance, Strider [?], [?] has been presented using the Spark Streaming and Kafka framework. The DRSS [?] utilizes the Apache Storm platform. One of the challenges with these approaches is the complexity of these solutions, which leads to difficulties in their utilization, implementation, or upgrades.

2. Centralized models that lack high processing capacity support. These approaches suffer from limitations such as collaboration, sharing, expressiveness, and scalability [?]. Additionally, according to [?], these centralized RSP approaches face challenges such as high scalability issues, difficulties in executing concurrent queries, and increasing input load. Centralized RDF stream processing methods include C-SPARQL [?], SparqlStream [?], SPARKWAVE [?], and CQELS [?].

In [?], the MAS4MEAN framework, based on a multi-agent model, has been proposed to address the limitations of centralized approaches. In this article, multi-agent systems are employed in RSPs to mitigate scalability issues and implementation complexities. This approach utilizes three types of agents: sensing agents, processing agents, and reasoning agents. Within the processing agents, multiple instances of the C-SPARQL engine are utilized for parallel query processing. Consequently, the number of events assigned to each C-SPARQL engine is reduced, enabling the MAS4MEAN method to handle high volumes of events efficiently.

However, in this approach [?], query acceleration for complex queries is not feasible, as each query is executed by a single agent. Moreover, preventing the execution of redundant subqueries or optimizing queries is not possible. The only enhancement provided in this article is the distribution of events among different C-SPARQL engines by each agent, offering multiple C-SPARQL engines for executing various queries. Additionally, there is no provision for executing queries locally alongside the data. Data must be transmitted from sensors to processing agents, leading to suboptimal network bandwidth utilization and query execution delays. Consequently, as the volume of data required for query execution increases and queries become more complex, the query execution time will further increase.

So far, researchers have proposed a set of operators for executing continuous queries in SPARQL, covering heterogeneity issues from data streams. However, paralleliza-

Table 3: Summary of related works

Method	Query Decomposition	Duplicate Query Execute	Handle Large Data Stream	Query Processing	Data Type	Layered Architecture	Query Optimization
Zafeiropoulos et al. 2008 [?]	✗	-	✗	C	RDF	✓	✗
Patni et al. 2011 [?]	✗	-	-	C	RDF	✗	✗
De et al. 2012 [?]	✗	-	-	C	RDF	✗	✗
Phuoc et al. 2012 [?]	✗	-	-	C	RDF	-	✗
Gyrard et al. 2013 [?]	-	-	-	C	RDF	✗	-
Nagib et al. 2016 [?]	✗	-	✗	C	RDF	✗	✗
Dastjerdi et al. 2016 [?]	-	-	-	C	Non-RDF	✓	-
Giordano et al. 2016 [?]	-	-	-	-	Non-RDF	✓	-
Khrouf et al. 2016 [?]	✗	✓	✓	D	RDF	✗	✗
Calbimonte et al. 2016 [?]	Syntactically	✓	✓	C	RDF	✗	✗
Wang et al. 2017 [?]	✗	-	-	C	RDF	✓	✗
Arkian et al. 2017 [?]	✗	-	✓	D	Non-RDF	✓	✗
Dia et al. 2018 [?]	Syntactically	✗	✓	D	RDF	✗	✓
Tuli et al. 2019 [?]	✗	-	✓	D	Non-RDF	✓	✗
Cao et al. 2019 [?]	✗	✓	✓	D	Non-RDF	✓	✗
Al-Baltah et al. 2020 [?]	✗	-	-	C	RDF	-	✗
Mebrek et al. 2020 [?]	✗	✓	✓	D	RDF	✓	✗
Ortiz et al. 2022 [?]	✗	-	-	D	Non-RDF	✓	✗
Bonte et al. 2023 [?]	Syntactically	-	✓	D	RDF	✓	✓
DISIF (our solution)	Semantically	✗	✓	D	RDF	✓	✓

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

tion and scalability aspects in RDF stream processing are not addressed. Non-distributed versions of SPARQL include Streaming SPARQL [?], C-SPARQL [?], SPARQL Stream [?], CQELS [?], EP-SPARQL [?], and SPARK-WAVE [?].

More precisely, as stated in [?],[?], existing methods encounter challenges in many scenarios such as executing a large number of concurrent queries, handling queries for large data, and wide range of data generation frequency. Therefore, various methods like DIONYSUS [?], CQELS Cloud [?], and C-SPARQL on S4 [?] have been introduced for real-time processing of large RDF data streams in a parallel and distributed manner. To efficiently distribute and compute large-scale RDF data streams, there is a need for effective partitioning of queries and data among multiple nodes while minimizing data exchange between them.

The [?] presents a distributed and scalable approach for processing RDF streams. The aim of this article is to decompose queries into subqueries and execute each subquery on parallel machines, minimizing data exchange between nodes. In [?], a distributed RDF SPARQL streaming is presented for scalable processing of C-SPARQL queries. This approach utilizes three components: query rewriting, query partitioning, and RDF graph partitioning. In the query rewriting phase, when multiple queries enter the system, it identifies common substructures among queries and prevents redundant execution of these sub-structures. These queries are then rewritten into a single query, ensuring that the sub-structures are executed only once. Subsequently, this rewritten query is sent to the Query Partitioning module.

In the Query Partitioner component, SPARQL queries are partitioned based on joins into partitions, and each partition is executed by a set of machines without the need for communication between them, thereby achieving query execution parallelization. Processing nodes in [?] are all identical, and there is no differentiation between nodes. Additionally, the assignment of tasks to nodes is not considered. The usage of master and worker nodes is also not employed, and how query execution distribute be-

tween nodes and how processing distribute between nodes are not explained.

The Waves method [?] offers a distributed approach to RSP systems using the Apache Storm framework. None of the centralized approaches can handle massive streaming data and scalability issues as they are designed to run on a single machine. To address and overcome scalability issues, the Waves approach [?] is presented as a distributed RSP system on big data frameworks like Apache Storm.

In [?] approach, the aim is to distribute the execution of queries by C-SPARQL across different nodes. However, this approach does not utilize query decomposition into subqueries, thus cannot prevent the execution of redundant subqueries and results in reducing system efficiency. Additionally, this approach does not improve the execution speed of individual queries, and is only suitable for processing large volumes of data. In other words, the query execution method remains the same, without any optimization for individual queries; the emphasis is solely on handling large data streams.

The StreamQR [?] method offers a query rewriting approach by rewriting user queries based on the ontology. In this approach, the user query is transformed into a Union of Conjunctive Queries (UCQ). The where clause of the C-SPARQL query changes according to the ontology, and some conditions are added to it, expanding the query. In other words, StreamQR [?] provides a query rewriting approach that can directly inject domain knowledge into the query, resulting in the generation of very large queries with numerous unions, which may be costly in terms of execution.

StreamQR enables the execution of continuous queries under entailment by rewriting the queries into multiple parallel queries. In this approach, the rewriting of the query does not involve adding a time window length or conditions such as Having, etc., and it is a simple form of query expansion. The actions performed in the StreamQR approach for query rewriting include: no changes are made to the predicates (only adding some predicates), no changes are made to the where conditions, it is only used for unionizing predicates, and there are no changes made

to the time window.

In Table 3, the "Query Decomposition" section indicates which of the mentioned frameworks can break down high-level queries into sub-queries for optimized execution. The term "syntactically" refers to a method where the query is divided into separate parts based on its structure (syntax). In contrast to the syntactic method, the "semantically" approach relies on various ontologies and concepts to translate the query into sub-queries.

The "Duplicate Query Execute" section examines the capability to prevent the execution of redundant queries, thereby enhancing system efficiency.

The "Layered Architecture" section investigates the presence of a layered architecture in the respective framework. In the absence of a layered architecture, the framework cannot leverage the benefits of edge nodes, fog nodes, and cloud nodes. In a flat architecture, nodes are homogeneous, leading to weaker resource management and system performance optimization compared to a layered architecture.

With an emphasis on dynamic adaptation, the DIVIDE platform [?] addresses the challenge of dynamically adapting IoT stream processing queries to real-time environmental contexts using Semantic Web technologies. By incorporating edge device monitoring and a cascading architecture, DIVIDE effectively manages context-aware query distribution and configuration. Additionally, the implementation of a Meta Model in DIVIDE enables dynamic query adjustments, enhancing performance under varying device and network conditions.

3. DiSIF: Distributed Semantic Information Fusion framework

In this section, we introduce the distributed version of semantic JDL model within the framework of a three-layer architecture that includes edge, fog and cloud layers. As case study, we examine this model on traffic detection problem.

3.1. Overview of DiSIF Framework

This section introduces the DiSIF framework, which consists of three layers: edge, fog, and cloud, designed for making time-sensitive and complex dependent decisions in IoT applications. As case study, we examine this model on traffic detection problem.

Next, the DiSIF framework's layers are examined from two perspectives: the JDL model and the fusion model.

3.1.1. The JDL model perspective

As depicted in Figure 4, the DiSIF framework consists of three layers: edge, fog, and cloud. Each layer comprises two types of nodes: worker nodes and masters. Worker nodes are responsible for receiving and processing data/streams from the surrounding environment or

lower layers and sending results to the corresponding master node.

In the edge layer of the DiSIF framework, decisions are made at the lowest level of processing (level one/ object refinement) in a real-time and resource-efficient manner, utilizing the nodes with minimal resources. The processes in this layer fall into the object refinement component of the JDL model.

Subsequently, the processes and decisions made at this layer are sent to the fog layer nodes for further processing and decision-making at a higher level (level two/ situation refinement). These decisions may involve detecting traffic events or congestion events on the streets.

In the fog layer, using the cityOnto ontology, decisions from the edge layer (level one) are combined to make new decisions at the fog layer (level two). For making large-scale decisions at level three (city level/ threat refinement), the results from the fog layer are sent to the cloud layer node. In this node, all information and decisions from lower layers are stored in a database, and heavy processing operations (threat refinement), including predicting traffic based on vast data volumes and implementing managerial decisions at the city level, are performed. Therefore, information processing, fusion, and decision-making occur layer by layer.

3.1.2. The fusion model perspective

In this section, we first address the definition of the concepts of independent queries and dependent queries.

- **Independent Query:**

An independent query in C-SPARQL is a query that can be executed without dependence on the results of other queries. Additionally, it can be decomposed into independent subqueries, which can be executed in parallel without any interdependency, enabling efficient and scalable data processing. The results of these subqueries are separate from each other and can be executed and obtained without any connection to each other. An example of such a query is to investigate the number of vehicles present in a specific area. This query can be divided into subqueries that count the number of vehicles on the streets within that area. Each subquery can be executed independently, and the results can be published separately.

- **Dependent Query:**

A dependent query in C-SPARQL is a query that relies on the results of one or more previous C-SPARQL queries for its execution. Dependent C-SPARQL queries are usually used to perform operations such as analysis or further processing on the results of previous queries. This is essential for complex data analysis tasks involving sequential or fusion operations on RDF data streams.

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



C:/ArticleLatexImagesFinal/AllinOneFrameWork_modified.drawio.pdf

Figure 2: DiSIF framework

- **Data Level**

The designed framework is structured to manage data across distinct layers, transitioning from the edge layer to the cloud layer. The nature of the data becomes increasingly abstract, and its contextual complexity rises. At the edge layer, the data primarily comprises sensor data (Level one). In the fog layer, the data transforms into processed data or information streams. Queries performed at the edge layer focus on sensor data and involve operations related to data fusion. As queries progress to higher levels, they address broader concepts, such as traffic patterns, utilizing nodes in the fog layer for information fusion.

- **Independent/Dependent Queries**

Queries defined in the DiSIF framework can be categorized into independent and dependent queries. Independent queries are usually inter-layer queries that can be executed simultaneously across multiple nodes, with their results combined afterward. These queries are assigned from worker nodes in the upper layer to master nodes in the lower layer.

Dependent queries, on the other hand, are intra-layer queries that must be executed sequentially and consecutively. The results of one query are considered as input for another query. These queries are executed between worker and master nodes within a layer.

- **Vertical vs. Horizontal Fusion**

Vertical fusion operations, resulting in an increase in the scale of the data. In other words, the input and

output of vertical fusion represent the same concept, differing only in size and scale. For example, traffic data from multiple streets can be fused to generate traffic data for an entire area.

Horizontal fusion, results in a change in the concept of the data. For instance, it involves fusing congestion data from a street to generate traffic data specific to that street. In this type of fusion, only the concept of the output data changes, while the scale and size of the data remain unchanged. Horizontal fusion can occur within each layer and between master and worker nodes.

- **JDL Model Components**

In the DiSIF framework, the edge layer handles operations related to the JDL object refinement component, while situation refinement can be performed in the fog layer on the data received from the edge layer. Similarly, in the cloud layer, threat refinement operations can take place.

The subsequent section presents an overview of each layer in the DiSIF framework, aiming to deploy a Distributed Semantic JDL model.

3.2. Edge Layer

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

In the Edge Layer, processing and fusion operations are performed on the streams of data from sensors. In the worker node, operations such as receiving data from

C:/ArticleLatexImagesFinal/Overall_Framework_Communication_OK.drawio.pdf

Figure 3: DiSIF Communication

C:/ArticleLatexImagesFinal/h3.drawio.pdf

Figure 4: The JDL and fusion model perspectives

sensors, executing queries assigned by the master node, and sending result streams back to the master are carried out. In the master node, operations involve receiving user queries via UCI, storing data in the database, registering and retrieving information about worker nodes in the NodePlatform, executing and assigning queries to worker nodes, receiving and aggregating received streams from worker nodes.

NodePlatform

Node	Concept	Location	Master
N_1^w	<i>Congestion</i>	<i>loc1</i>	N_1^m
N_1^w	<i>Congestion</i>	<i>loc2</i>	N_1^m
N_2^w	<i>Traffic</i>	<i>loc3</i>	N_3^m
N_3^w	<i>Congestion</i>	<i>loc5</i>	N_2^m

Table 4: Example of NodePlatform

The Node Platform as shown in Table 4 facilitates the registration of worker nodes with their corresponding master nodes, capturing essential information for each worker node such as its name, the concepts it generates, the locations or streams it can produce along with its corresponding master node. It is instrumental in breaking down queries into sub-queries, dispatching these sub-queries to

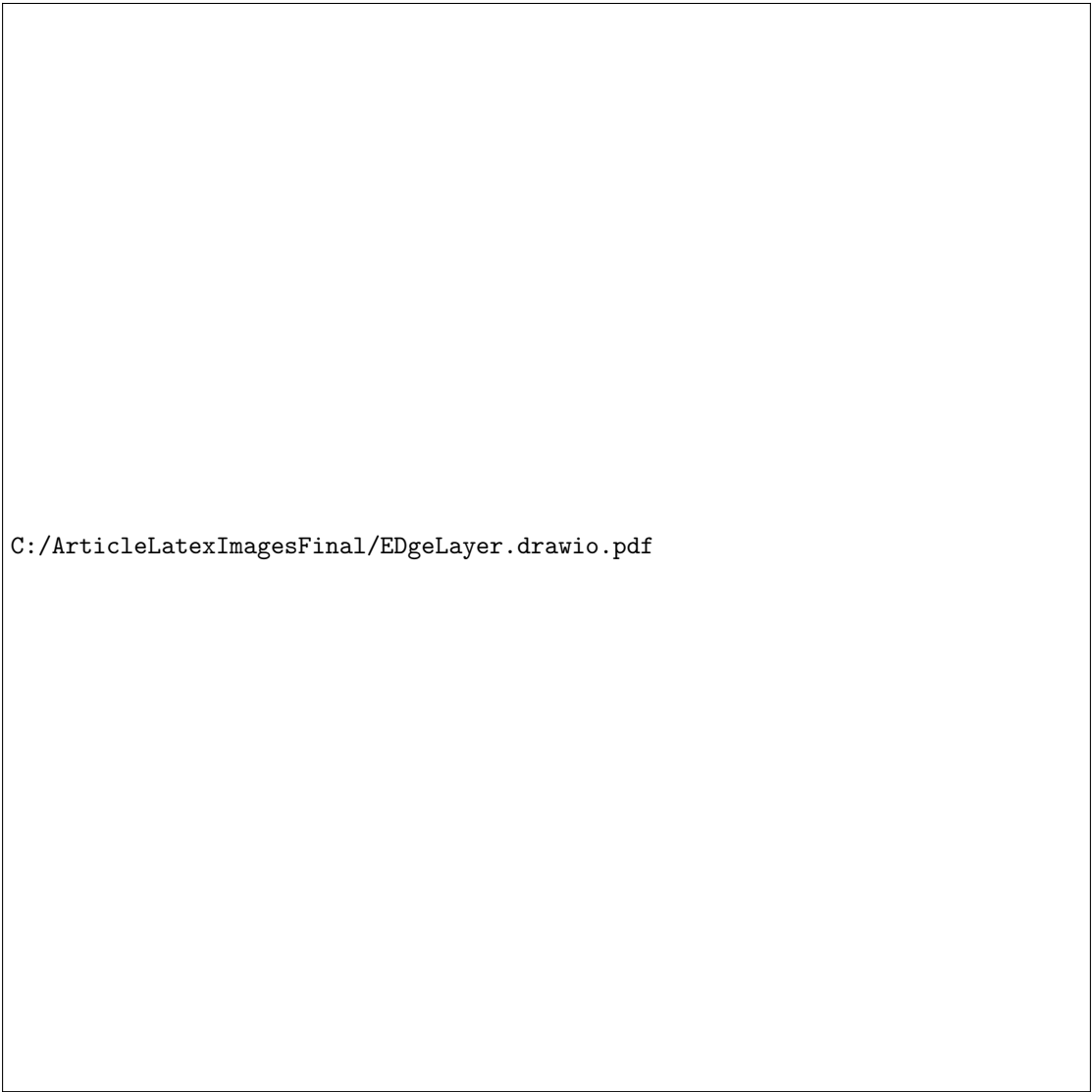
the appropriate worker nodes, and receiving result streams back at the master node.

Within the master node, the NodePlatform serves as a repository for information about registered worker nodes, allowing for easy retrieval and discovery of nodes associated with specific concept and location.

This functionality ensures that queries are directed to the correct node that generating specific concept for corresponding location.

The database component is used to store data and received streams from worker nodes. The stored data is utilized for query execution in the master node. The UCI component receives queries from users and sends them to the master node. It also returns the results of query execution from the master node to the user. Therefore, all user operations are performed through UCI. In the master node, horizontal fusion operations on data (object refinement) from the Edge layer are conducted. Ultimately, the generated streams are sent to the worker nodes in the Fog layer. Algorithm 1 outlines the assignment procedure of a worker node to the master node.

The metadata information sent from the worker node to the master node is contained: the name of *workerNode*, set of concepts that the *workerNode* can generate, the locations/streams it can produce along with its corresponding master node.



C:/ArticleLatexImagesFinal/EDgeLayer.drawio.pdf

Figure 5: The Edge Layer

Algorithm 1 RegisterWorkerNode

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

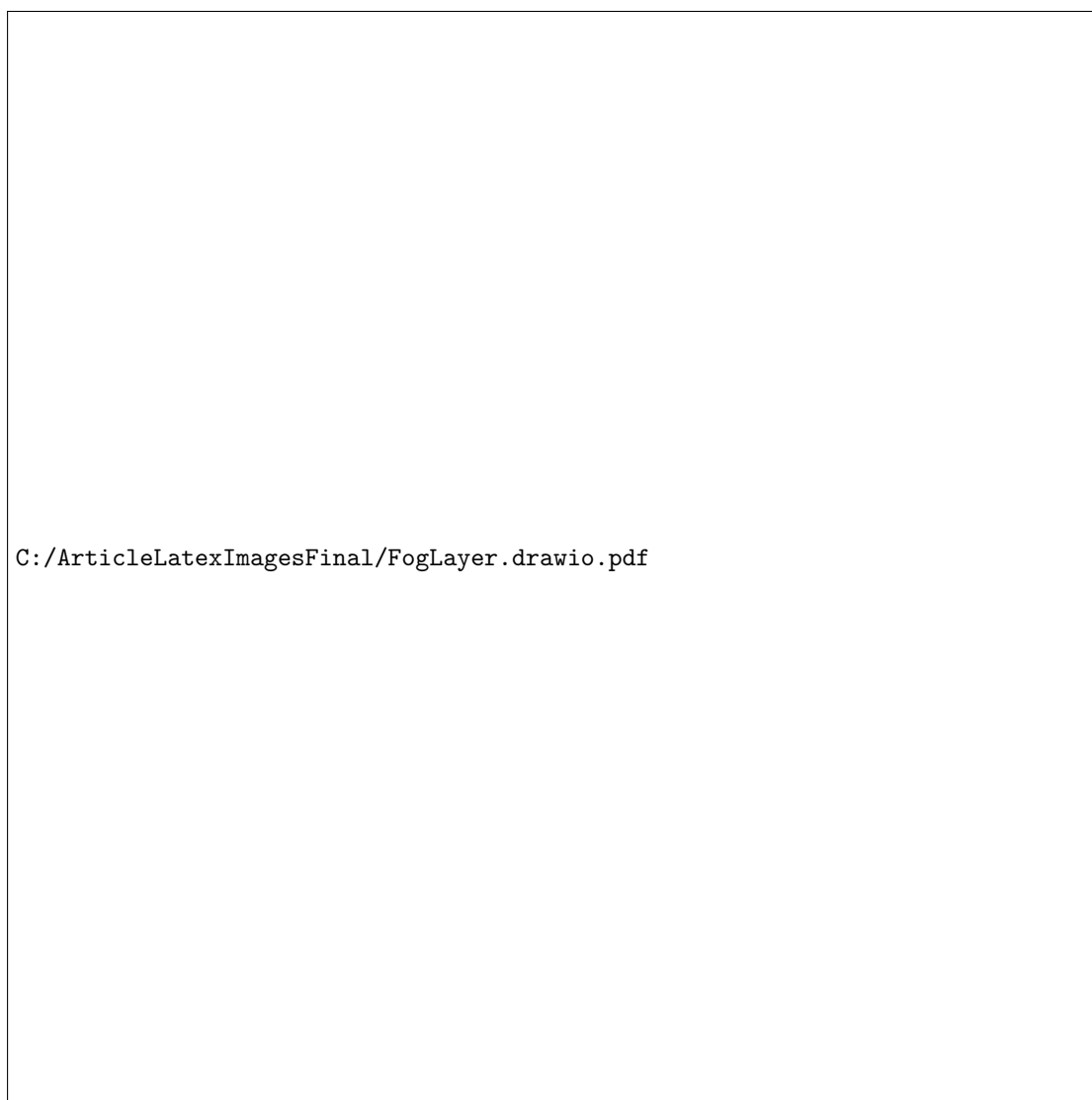


Figure 6: The Fog Layer

3.3. Fog layer

An overall view of the DiSIF framework for the Fog layer is illustrated in Figure 6. In the Fog layer, the data in the form of concept streams (instead of data stream) are processed and fused. Similar to the Edge layer, the operations of a worker node in the Fog layer include receiving concept streams from the Edge Layer and executing assigned queries from the master node.

In the master node of the Fog layer, operations involve receiving user queries via UCI, storing data in the database, registering and retrieving information about worker nodes in the NodePlatform, executing and assigning queries to worker nodes, receiving data streams from worker nodes, and fusing received streams from worker nodes.

The components in this layer are similar to those in the Edge Layer. For vertical fusion in the fog layer, particularly in the context of intelligent traffic, the cityOnto ontology is essential. In this layer, this ontology serves as external data.

In the master node, horizontal fusion enables the completion of situation refinement operations, and the resulting streams are forwarded to the cloud layer for extensive processing.

Algorithm 4 outlines the query response process within the master node. In this algorithm, the operations performed on the master node for processing user queries. It handles two types of queries: SPARQL queries (line 4) for answering static queries based on the data in the database (DB), and C-SPARQL queries (line 6) for responding to queries over data streams. In line 7, the algorithm checks whether the concepts used in the query have already been registered by any node within the NodePlatform. If the corresponding node exists, in line 8, the location/stream of that node is compared with the one in the query. If the required locations in the query have been previously registered for that node, the query is divided into sub-queries based on the different locations (if necessary), and each sub-query is forwarded to the corresponding nodes. The results are then gathered and aggregated by the master node (lines 9 to 11).

In Algorithm 2, if the corresponding node for the concept and location extracted from the user's query is found, the operations for retrieving data from that node (line 6), executing the query, and returning the results (lines 7–8) are carried out. If node not found, in (line 10), one of the nodes that have been previously defined in the NodePlatform for the corresponding master node and for the requested location/stream is selected. Then, in lines 11 and 12, based on the ConceptOnto ontology, all the necessary elements for constructing a query to generate the new concept are extracted, and in line 13, this query is generated using Algorithm 3. Finally, in line 14, the NodePlatform is updated to include the new concept for the node and the specified location, and in line 15, the constructed query is sent to the selected node for execution.

3.4. Cloud layer

In the cloud layer, considering the received streams from the fog layer, semantic fusion operations on the streams take place. Threat refinement at a macro level are conducted in this layer. Global concept streams received from the fog layer are stored in a database, where predictions are made based on this data. In the cloud layer, heavy processing is performed periodically or as needed on historical data, with the results reported to the user. One significant task in this context is traffic prediction, which forecasts traffic in various locations at a macro level using both the stored data and ongoing streams from the fog layer.

3.5. Query Execution Request Flow

As depicted in Figure 7, one FogWorker and two EdgeMaster nodes, labeled *FogWorker1*, *EdgeMaster1*, and *EdgeMaster2*, are present. Initially, edge worker nodes perform the registration and storage of their metadata in the NodePlatform of the EdgeMasters. In this manner, all streams that these worker nodes can generate, along with the concepts (query predicates) they can execute, are stored in the EdgeMaster nodes.

A query is submitted through *FogWorker1*, which refers to two independent subqueries, *Traffic(A1)* and *Traffic(A2)*, derived from the CityOnto ontology. These subqueries are executed in parallel on two separate locations/streams, *A1* and *A2*, using *EdgeMaster1* and *EdgeMaster2*. In the EdgeMaster nodes, the concept generation component (Algorithm 3) utilizes the ConceptOnto ontology (Figure 8) to generate concepts that do not have incoming streams. This process produces dependent subqueries, such as *Congestion(a1)* and *Congestion(b1)*. These sub-queries are then executed on edge worker nodes, and the results are returned to the EdgeMaster node.

Horizontal fusion is performed in the Fuse results component of EdgeMasters nodes, and the results are referred back to *FogWorker1*. Within the Fuse results component from *FogWorker1*, vertical fusion is carried out combining the outputs of queries on streams *A1* and *A2*, resulting in the final output.

4. Evaluation

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

C:/ArticleLatexImagesFinal/Communication_new.drawio.pdf

Figure 7: Query execution request flow

Algorithm 2 Process Query

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

C:/ArticleLatexImagesFinal/ConceptOnto_GraphViz_new.pdf

Figure 8: ConceptOntology

Algorithm 3 Construct Query

```
1: procedure CONSTRUCTQUERY(Concept, Location, Entity, Range, hasAddition, ConceptNew,  
   filterParam, filterConstraints, hasGroup, hasHaving, hasOrder)  
2:   query=  
     CONSTRUCT {?l concept:{Concept} ?a }  
3:   FROM STREAM {Location} [RANGE {Range}]  
4:   WHERE { {Entity} concept:{ConceptNew} {filterParam},  
5:   {Entity} {hasAddition} ,  
6:   FILTER ({filterConstraints})  
7:   }  
8:   GROUP BY {hasGroup}  
9:   HAVING {hasHaving}  
10:  ORDER BY {hasOrder}  
11: end procedure
```

Algorithm 4 Query response in masterNode

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

As mentioned, in the centralized JDL fusion model, generating the desired outputs requires collecting all the necessary data in a central node, where queries and corresponding components are executed on these aggregated data to produce the outputs. In contrast, in the distributed JDL approach, there is no need to send all data to a central node. Instead, by distributing query processing, only the query execution results are sent to other nodes.

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

- **Network Load**

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

- **Execution Time of Dependent Queries**

In the JDL model, the situation refinement component, requiring the use of outputs from the object refinement component, executes dependent queries for output generation. As subqueries need to be executed first to provide the necessary input for dependent queries, the time to produce outputs for dependent queries increases. In the centralized approach, both subqueries and dependent queries are executed on a single node, while in the distributed JDL approach,

subqueries are executed on worker nodes, and dependent queries are executed on the master node. Consequently, the execution time of dependent queries is reduced, making it more efficient compared to the centralized approach.

- **Execution Time of Independent Queries**

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

- **Memory Consumption**

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

- **Data Security**

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

C:/ArticleLatexImagesFinal/result_TwoMaster_OneMaster.pdf

Figure 9: Object refinement performance: independent query execution time for different master nodes

can be performed locally on local data, with the results sent to the master node. Thus, the security issue related to data transfer is mitigated in this distributed approach.

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

4.1. Object refinement performance in the Edge layer

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

4.1.1. Centralized and Distributed approaches

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

In Figure 9, the results of executing various queries in the centralized scenario, where only one edge master node exists and for distributed approach with two edge master nodes, are presented. In this case, the execution time of queries is analyzed for different stream rates. As observed, for query Q1, the execution time experiences a sudden increase at a stream rate of 30000 triples/s. Similarly, query Q2 shows a sudden increase at a stream rate of 15000 triples/s, Q3 at 5000 triples/s, and finally, Q4 at 3000 triples/s.

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

As depicted in Figure 9, the execution time for different stream rates is approximately halved. The reason for this slight increase in the execution time is that a short period is spent aggregating the results from these two nodes. Therefore, in comparison to the centralized approach for executing independent queries, this distributed scenario exhibits lower execution times.

4.2. Situation refinement performance in the Fog layer

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

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

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

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

Traffic Discovery Scenario

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

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

Congestion Detection Scenario

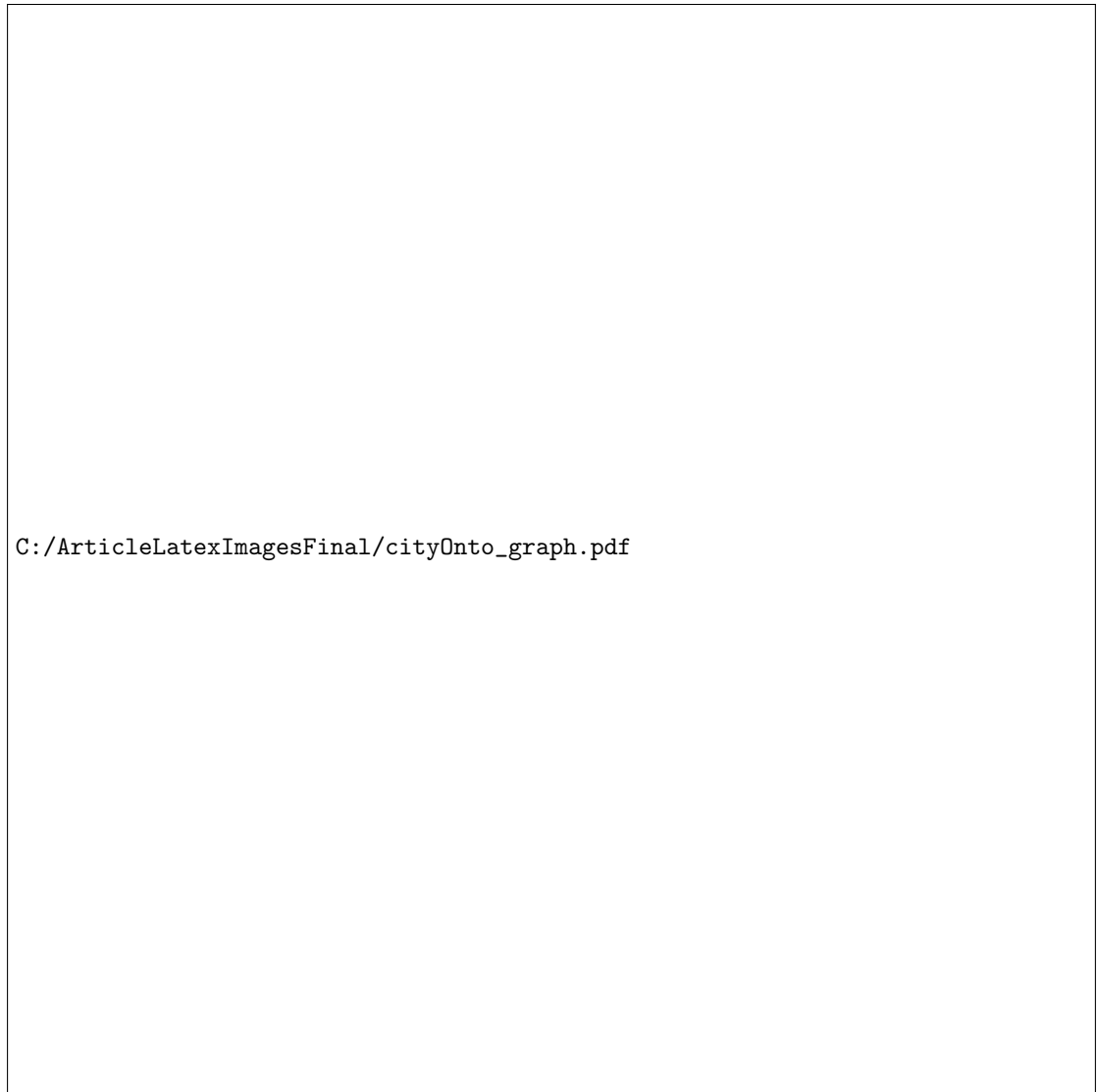
To evaluate the performance of both centralized and distributed JDL approaches for congestion detection, we employ various queries with diverse complexities as detailed in the Appendix A as Queries Q_1, Q_2, Q_3, Q_4 .

Query Q_1

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

Query Q_2

Continuing with the congestion detection scenario, Query Q_2 identifies regions where at least three vehicles have speeds below 50 km/h, indicating congestion. The



C:/ArticleLatexImagesFinal/cityOnto_graph.pdf

Figure 10: CityOnto example

results are then sorted in ascending order based on location. This query establishes a more specific criterion for detecting congestion by taking into account both the speed condition and the minimum number of vehicles present in a given area.

Query Q_3

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

Query Q_4

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

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

4.2.1. Query execution time

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

- Centralized approach

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

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

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

- Distributed approach

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

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

As can be observed from the equations, in the centralized approach, the execution time increases linearly with the increase in the number of queries Q_i . This is due to the sequential processing. In the distributed

approach, the queries are processed in parallel, which reduces the execution time to the maximum time required to execute any of the Q_i queries. Therefore, for a large number of requests N , the distributed approach significantly reduces the execution time compared to the centralized approach due to parallel execution.

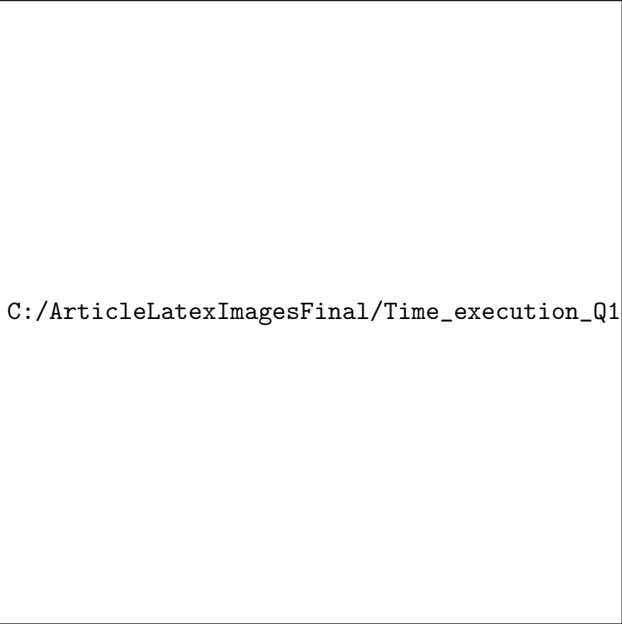
To analyze the execution time for queries Q_1 to Q_4 and subsequently query Q_m , Figure 11 illustrates the results for various data transmission rates (triples per second). This analysis compares both the centralized JDL (C-JDL), StreamQR and DiSIF approaches.

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

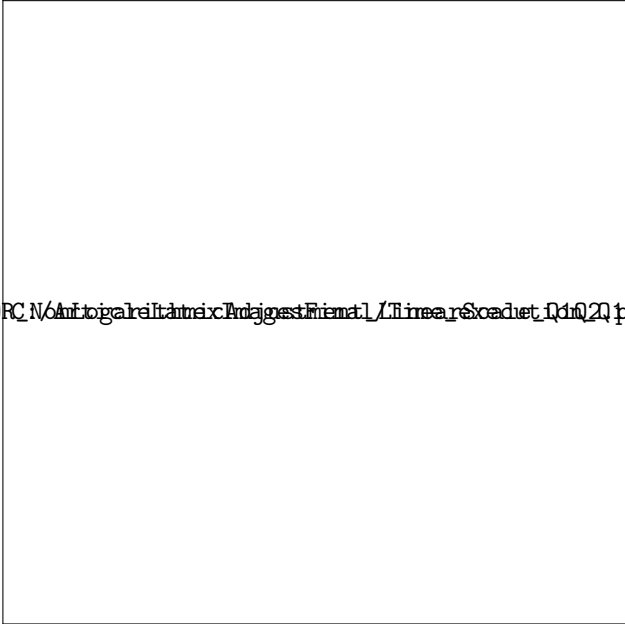
In the StreamQR model, queries are aggregated and executed as a single large query, which significantly improves execution time compared to the C-JDL method. Since all queries are aggregated and executed in one process, the sequential execution is eliminated, and processing speed increases. However,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

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

As observed in Figure 11a and 11b, the execution time of DiSIF(Q_1), StreamQR(Q_1) and C-JDL(Q_1) increases almost linearly with the increase in the data sending rate. Moreover, the time needed for executing DiSIF(Q_1) is generally less compared to others, and



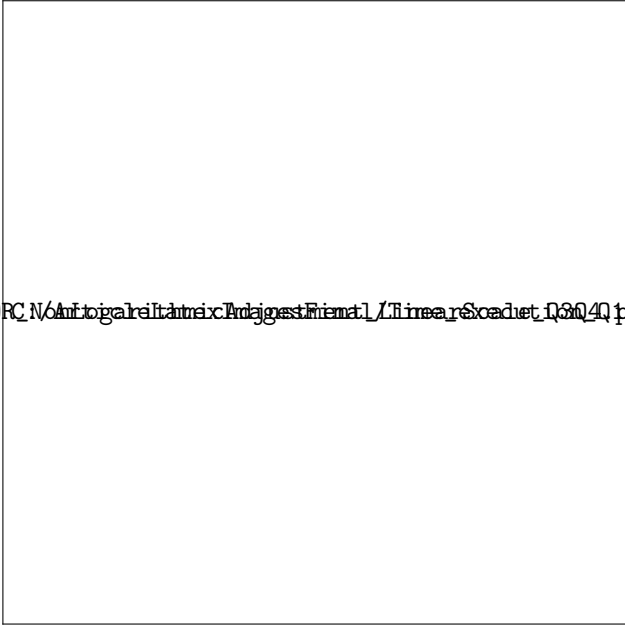
(a) Comparing Execution Times for Q1 and Q2 in Centralized and Distributed Environments



(b) Comparing Logarithmic Execution Times for Q1 and Q2 in Centralized and Distributed Environments



(c) Comparing Execution Times for Q3 and Q4 in Centralized and Distributed Environments



(d) Comparing Logarithmic Execution Times for Q3 and Q4 in Centralized and Distributed Environments

Figure 11: Query execution times in Centralized and Distributed Environments

this time difference remains approximately constant across various data sending rates.

Figures 11c and 11d reveal an exponential increase in execution time for queries C-JDL(Q3), StreamQR(Q3), C-JDL(Q4) and StreamQR(Q4) as the data sending rate rises. In contrast, the execution time for DiSIF(Q3) and DiSIF(Q4) shows a much less significant growth rate. This discrepancy is due to the dependency of query Q_m on Q_3 and Q_4 . When both queries are executed on the master node (centralized mode), the delay becomes substantially higher. In the DiSIF model, however, queries Q_3 and Q_4 are processed on the worker nodes, and only the results are sent to the master node, thereby reducing delays associated with producing results for Q_m . This illustrates the stability or robustness of the DiSIF method.

4.3. Network load perspective

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

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

$$L_c = \sum_{i \in N} D_i \quad (3)$$

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

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

$$L_d = \sum_{i \in N} R_i \quad (4)$$

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

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

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

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

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

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

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

```
"7103 congestion"
```

This message contains 15 characters and 14 bytes. In Figure 12, the network usage for sending messages for various queries in C-JDL, StreamQR and DiSIF approaches is illustrated.

As observed in Figure 12, the volume of messages sent over the network with Kafka in the C-JDL/StreamQR approaches is significantly higher compared to DiSIF approach.

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

Furthermore, as the data streaming rate increases, the number of messages sent also rises, highlighting the distinction between C-JDL/StreamQR approaches and DiSIF. Additionally, as the complexity of the queries increases (Q_{1i} Q_{2i} Q_{3i} Q_{4i}), fewer messages are sent in the network. In contrast, with C-JDL/StreamQR approaches, there is no significant difference in the volume of sent messages with respect to the complexity of the queries.

4.4. Memory Consumption Perspective

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

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

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

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

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

However, in the distributed approach, since queries Q_i and Q_m are executed in parallel on different nodes, the

C:/ArticleLatexImagesFinal/Network_Usage_vs_Log_vs_Triples.pdf

Figure 12: Network usage comparison for different stream rates

memory consumption is equal to the maximum of the memory requirements for Q_i and Q_m in both worker and master nodes.

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

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

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

As shown in Figure 13a, memory consumption for executing queries Q_1 and Q_2 in the C-JDL is over four times greater than in the DiSIF. This occurs because, in the C-JDL, Q_1 (or Q_2) must be processed to generate outputs stored in the master node's memory, which are then used as input for query Q_m to obtain the final results. As a result, memory usage in the C-JDL is significantly higher compared to the DiSIF.

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

On the other hand, Figure 13c shows that memory consumption for queries Q_3 and Q_4 in C-JDL and StreamQR is significantly higher than in DiSIF approach. This increase is due to the use of AVG functions and CONTAINS, which can elevate memory usage. Storing the outputs in memory and then executing Q_m on these outputs substantially increases memory consumption for Q_3 and Q_4 in C-JDL and StreamQR compared to DiSIF approach.

Moreover, Figures 13a and 13c demonstrate that the AVG and CONTAINS functions in queries Q_3 and Q_4 can significantly increase memory consumption, particularly when data transmission rates exceed 40,000 triples per second in C-JDL and StreamQR. Additionally, as the data transmission rate increases, memory consumption for Q_4 surpasses that of Q_3 , a trend that becomes noticeable when data transmission rates surpass 40,000 triples per second.

5. Conclusion and future works

This study introduces a novel distributed semantic JDL fusion model tailored for smart city applications, leveraging a three-layer architecture consisting of edge, fog,

and cloud layers. Our framework addresses the limitations of centralized fusion models, particularly the inefficiencies associated with processing vast volumes of heterogeneous data in smart cities. Key benefits of our approach include:

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

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

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

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

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

Furthermore, the DiSIF framework supports real-time, decentralized decision-making, which is critical for addressing the diverse and dynamic needs of urban environments. The innovative approach of combining data fusion at different layers with a distributed query execution model provides a comprehensive solution for the complex data management challenges faced by smart cities.

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

6. Declaration of competing interest

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

Appendix A. Queries

Query Q_m

REGISTER QUERY Traffic AS

```
PREFIX ex: <http://myexample.org/>
PREFIX loc: <https://location.com/>
PREFIX stat: <https://status.com/>
```



(a) A single stream receiver node



(b) A single stream receiver node(log)



(c) A single stream receiver node



(d) A single stream receiver node (log)

Figure 13: Memory consumption for different stream receiver nodes

```
PREFIX cnt: <https://cntVehicles/>
PREFIX concept: <https://concept.com/>
```

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

Query 1

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

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

Query 2:

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

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

Query 3:

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

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

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

Query 4:

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

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

    FILTER(?speed < 50)
    FILTER (CONTAINS(str(?location), "2") ||
        CONTAINS(str(?location), "3") ||
        CONTAINS(str(?location), "1"))
}
UNION
{
```

```

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

```

Mohsen Kahani is a professor of computer engineering, IT director and head of Web Technology Laboratory at Ferdowsi University of Mashhad and visiting Researcher at Cloud Computing and Distributed Systems (CLOUDS) Laboratory, School of Computing and Information Systems, The University of Melbourne, Australia. His research interests include semantic web, software engineering, natural language processing and process mining.

Ramin Rezvani-Khorashadizadeh is a Ph.D. candidate at the Department of Computer Engineering, Ferdowsi University of Mashhad, Iran. His research focuses on areas such as RDF stream processing, smart city applications, big data analysis, and fog computing. He is an active member of the Web Technology Laboratory (WTLab), where he works on developing cutting-edge technologies for data processing and decision-making in smart cities. Ramin's work aims to address challenges in handling large, heterogeneous data streams, leveraging semantic data models and distributed computing frameworks.



Mohsen Kahani




Ramin Rezvani-Khorashadizadeh



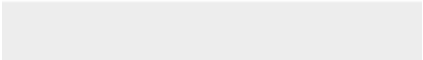



Click here to access/download
Supplementary Material
DiSIF_article.rar





Click here to access/download
Supplementary Material
Author_Photo.rar



Declaration of Interest Statement

The author, **Ramin Rezvani-Khorashadizadeh**, declares that there are no known competing financial interests or personal relationships that could have appeared to influence the work reported in this manuscript.