European Commission | Horizon 2020 European Union funding for Research & Innovation

Big Data technologies and extreme-scale analytics

Multimodal Extreme Scale Data Analytics for Smart Cities Environments

D5.1: MARVEL Minimum Viable Product[†]

Abstract: The purpose of this deliverable is to describe all activities related to the design, implementation, and release of the MARVEL Minimum Viable Product (MVP). This document sets the scope and goals of the MVP, as well as discusses the decisions to achieve these goals, by means of definition of specific use case scenarios, selection of MARVEL components, and infrastructure that facilitates the operation and demonstration of the use cases. More specifically, the document details the adjustments of the MARVEL architecture, the specific role of each component within the MVP, integration and deployment processes and tasks, and an extensive demonstration of the selected use cases from the user point of view. Finally, the MVP results are linked to MARVEL's objectives, as this release paves the way towards the first complete prototype of the MARVEL framework.

Contractual Date of Delivery	31/12/2021
Actual Date of Delivery	31/12/2021
Deliverable Security Class	Public
Editor	*Christos Dimou (ITML)*
Contributors	ITML, FORTH, IFAG, AU, ATOS, FBK, TAU, STS, UNS, GRN, ZELUS, PSNC
Quality Assurance	*Manolis Falelakis (INTRA)* *Tomás Pariente Lobo (ATOS)*

[†] *The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957337.*

The *MARVEL* Consortium

Part. No.	Participant organisation name	Participant Short Name	Role	Country
1	FOUNDATION FOR RESEARCH AND TECHNOLOGY HELLAS	FORTH	Coordinator	EL
2	INFINEON TECHNOLOGIES AG	IFAG	Principal Contractor	DE
3	AARHUS UNIVERSITET	AU	Principal Contractor	DK
4	ATOS SPAIN SA	ATOS	Principal Contractor	ES
5	CONSIGLIO NAZIONALE DELLE RICERCHE	CNR	Principal Contractor	IT
6	INTRASOFT INTERNATIONAL S.A.	INTRA	Principal Contractor	LU
7	FONDAZIONE BRUNO KESSLER	FBK	Principal Contractor	IT
8	AUDEERING GMBH	AUD	Principal Contractor	DE
9	TAMPERE UNIVERSITY	TAU	Principal Contractor	FI
10	PRIVANOVA SAS	PN	Principal Contractor	FR
11	SPHYNX TECHNOLOGY SOLUTIONS AG	STS	Principal Contractor	CH
12	COMUNE DI TRENTO	MT	Principal Contractor	IT
13	UNIVERZITET U NOVOM SADU FAKULTET TEHNICKIH NAUKA	UNS	Principal Contractor	RS
14	INFORMATION TECHNOLOGY FOR MARKET LEADERSHIP	ITML	Principal Contractor	EL
15	GREENROADS LIMITED	GRN	Principal Contractor	MT
16	ZELUS IKE	ZELUS	Principal Contractor	EL
17	INSTYTUT CHEMII BIOORGANICZNEJ POLSKIEJ AKADEMII NAUK	PSNC	Principal Contractor	PL

Document Revisions & Quality Assurance

Internal Reviewers

1. *Manolis Falelakis, (INTRA)*
2. *Tomas Pariente Lobo, (ATOS)*

Revisions

Version	Date	By	Overview
0.0.7	31/12/2021	Christos Dimou	PC and TC review comments
0.0.6	29/12/2021	Christos Dimou	Figure updates
0.0.5	27/12/2021	Christos Dimou	Fixed minor issues
0.0.4	24/12/2021	Christos Dimou	Revised document after internal reviews
0.0.3	07/12/2021	Christos Dimou	First full draft. Integration of inputs from partners
0.0.2	10/11/2021	Christos Dimou	Comments on the ToC.
0.0.1	03/11/2021	Christos Dimou	ToC.

Disclaimer

The work described in this document has been conducted within the MARVEL project. This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 957337. This document does not reflect the opinion of the European Union, and the European Union is not responsible for any use that might be made of the information contained therein.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

AI	Artificial Intelligence
API	Application Programming Interface
AV	Audio-Visual
AVCC	Audio Visual Crowd Counting
CI/CD	Continuous Integration / Continuous Delivery
DFB	Data Fusion Bus
DMP	Data Management Platform
DMT	Decision-Making Toolkit
DNS	Domain Name System
E2F2C	Edge-to-Fog-to-Cloud
EC	European Commission
FFMPEG	Fast Forward MPEG
GUI	Graphical User Interface
HDFS	Hadoop Distributed Files System
HLS	HTTP Live Streaming
HPC	High Performance Computing
HTTPS	Hypertext Transfer Protocol Secure
IoT	Internet of Things
IP	Internet Protocol
JSON	JavaScript Object Notation
KPI	Key Performance Indicator
MEMS	Micro Electro-Mechanical Systems
ML	Machine Learning
MQTT	Message Queuing Telemetry Transport
MVP	Minimum Viable Product
OLAP	Online Analytical Processing
PANN	Pretrained Audio Neural Networks
PDM	Pulse Density Modulation
PFLOPS	Peta Floating-Point Operations Per Second
POE	Power Over Ethernet
RAM	Random Access Memory
REST	Representational State Transfer
RTSP	Real-Time Streaming Protocol

SED	Sound Event Detection
SNR	Signal-to-Noise Ratio
SSO	Single Sign-On
TB	Terabyte
TRL	Technology Readiness Level
UI	User Interface
UJ	User Journey
URL	Uniform Resource Locator
USB	Universal Serial Bus
UX	User experience
VCS	Version Control System
VM	Virtual Machine
WP	Work Package

Executive Summary

This deliverable showcases all activities carried out towards the realisation of the MARVEL Minimum Viable Product (MVP). Following the *Lean start-up methodology* [1], the MARVEL project aims to the early release of a functional, end-to-end, minimum demonstrator that serves as a proof-of-concept for the overarching goals of MARVEL and displays the potentials of the sought solution. The deliverable has been developed within the scope of *'WP5 – Infrastructure Management and Integration'*, under Grant Agreement No. 957337.

The work presented in this document embarks from previously submitted deliverables *'D1.2 – MARVEL's Experimental protocol'* and *'D1.3 – Architecture definition for MARVEL framework'*, which define in detail the MARVEL pilot use cases and the refined framework architecture, respectively. The information drawn from these documents constitutes the foundations for the development of the MVP. We have selected use case *'GRN4: Junction Traffic Trajectory'*, implemented in the city of Malta, as a use case that is both mature in terms of technical requirements, and representative of the most fundamental aspects of MARVEL's framework. Additionally, we have proceeded with a careful selection of a subset of the MARVEL architecture, and a corresponding list of MARVEL components, that support the implementation and demonstration of the selected use case.

More specifically, for the selected use case, we have defined three different scenarios to cover and showcase different parts of the MARVEL architecture, including edge data transfer, AI components for event detection, cloud-level management, and aggregation of data, and, finally, presentation of the detected events to the end-user via a graphical web user interface.

In terms of technical details, this document provides a detailed presentation of the allocated infrastructure that supports the execution of the MVP, as well as a comprehensive description of participating components and technologies offered by MARVEL partners, including functionality, role of each component within the MVP, and component interfaces that facilitate communication and integration with other components.

In this deliverable, we discuss how the MVP addresses and contributes to specific WP5 and overall project goals, and its function as the foundation over which the two integrated prototypes of the framework are going to be developed, in later stages of the project (M18 and M30), that will, in turn, pave the way towards the final, large-scale deployment and operation of MARVEL into real-world settings (M36).

1 Introduction

1.1 Purpose and scope of this document

The purpose of this deliverable is to describe the scope, design rationale, technical details, and integration activities for the MARVEL Minimum Viable Product (MVP). Within the context of MARVEL, the MVP is an early release that serves as a proof-of-concept for the project's main objectives, as it offers a functional demonstration that is minimum but complete, in terms of end-to-end integration and delivery of value to the end-user.

In terms of design rationale, technical details and integration activities, this deliverable explains how the MARVEL consortium has selected a representative use case and defined a series of end-to-end scenarios that connect all layers of the MARVEL architecture, providing meaningful information on detected traffic events to the end-user. In technical terms, we have selected a series of components that were put in place and integrated into a solution that realises the purpose of the MVP, by interconnections that manage, process, and transfer data across all architectural layers in an efficient pipeline.

To complement the design and technical activities, this deliverable also lists the WP5 and project objectives that the MVP addresses and how it serves as a steppingstone towards the release of the more ambitious complete versions of the MARVEL framework in later stages of the project.

1.2 Intended readership

Deliverable D5.1 – 'MARVEL Minimum Viable Product' is a public document that accompanies the public demonstrator of this version of the MARVEL framework. The content found in this document aims to show all stakeholders and potential users of the framework, the feasibility of the MARVEL premise and the validity of the services provided to the end-users. Additionally, this document will serve as a guide to upcoming integrated releases of the MARVEL framework that will expand on the MVP in terms of expanding use cases, MARVEL components and technologies incorporated, and services offered.

1.3 Contribution to WP5 and project objectives

This deliverable has been composed within the context of *'WP5 – Infrastructure Management and Integration'*, and more specifically, it constitutes the first major output of *'Task 5.3 - Continuous integration towards MARVEL's framework realisation'*. The Description of Action (DoA) states the objectives of WP5 as such:

> *WP5 main objective is to ensure successful E2F2C framework delivery for distributed extreme-scale audio analytics. The framework allows for powerful, scalable, and real-time processing of multimodal audio-visual data on top of distributed deployment of MARVEL ML models. In details, WP5 aims to ensure: (i) provision and configuration of a powerful HPC infrastructure; (ii) orchestration of infrastructure resource management and optimised automatic usage of external computational and storage resources; (iii) seamless integration and quality assurance of software releases; (iv) quantifiable progress against societal, academic and industry validated benchmarks; (v) real-life experimentation and validation at large scale; and (vi) continuous alignment with the responsible AI planning and guidelines set out in T1.3.*

The MVP constitutes the first concrete step towards achieving the objectives of this work package, by providing a stripped-down, functioning version of the envisioned framework that, nevertheless, addresses and incorporates all mentioned attributes of this framework, including (i) provision of the required infrastructure, (ii) resource management, and (iii) integration of offered technologies. For specific aims (iv), (v), and (vi) mentioned above, the MVP provides the necessary foundation over which subsequent complete, large-scale versions of MARVEL will be further developed.

Specifically, for Task 5.3, the DoA states that:

> Task 5.3 *will implement and deploy the MARVEL integrated framework that realises the technology convergence defined in the MARVEL architecture specification T1.4. The MARVEL E2F2C Framework bridges Big Data technologies to IoT, Edge/Fog/Cloud computing and HPC, to allow real time decision making and monitoring of multimodal smart cities environments. This task realises societal and industrial opportunities in the smart-city domain (T6.2) by ensuring a smooth and effective integration of the separate MARVEL components. Aspects like interoperability, scalability, accountability, transparency, responsibility, and performance will be considered, as well as a continuous delivery approach, to achieve quality assurance in software releases. Releasing software frequently will not only respect the natural evolution of the technologies developed within the project but it will also allow developers to react promptly to the continuous data providers feedback. Reliability of these developments is also increased following this strategy as recurrent releases usually involve lesser and harmless changes. According to the plan (Sect.3.1.1), this task will describe in detail the MARVEL releases namely a proof-of-concept demonstration (Minimum Viable Product (MVP)) - M12, the 1st complete prototype – M18 and the 2nd Final prototype – M30.*

Being the first output of this task, the MVP described in this deliverable constitutes the first tangible solution that tackles most of the fundamental notions mentioned above and lays the foundations over which future releases will build upon. For the former case, the MVP is a first approach to technology convergence of various offerings, provides a demonstration of real-time interaction with the end-user, and provides an effective integration of the MARVEL components. Moreover, it has set the ground for the implementation of the described continuous integration/continuous delivery (CI/CD) approach. For the latter case, future versions of the MARVEL framework will provide the depth and scale required to put in place a complete CI/CD process, and address the mentioned interoperability, scalability, accountability, transparency, responsibility, and performance issues.

1.4 Relation to other WPs and deliverables

This deliverable relies on the foundational work conducted within '*WP1 – Setting the scene: Project setup'*. More specifically, the selection of Use Case for demonstration draws from the detailed material on Use Case descriptions of deliverable *'D1.2 – MARVEL's Experimental protocol*'. Additionally, deliverable '*D1.3 – Architecture definition for MARVEL framework*' is an important source for this work, as it contains the refined architecture, which is the blueprint for this release, as well as subsequent releases. D1.3 also provides useful information that D5.1 builds upon, as for example the description of available MARVEL components and their TRL, the grouping of components into building blocks that correspond to architectural layers, and the outline of integration processes that need to be applied.

This deliverable is also coupled with work in progress within the context of '*WP2 - MARVEL multimodal data Corpus-as-a-Service for smart cities', 'WP3 – AI-based distributed*

algorithms for multimodal perception and situational awareness', and *'WP4 - MARVEL E2F2C distributed ubiquitous computing framework'*. More specifically:

- From WP2, integral parts of the MVP are the Data management and distribution platform (Task 2.2), and the MARVEL Corpus-as-a-Service (Task 2.4).
- From WP3, the AI algorithms on Audio-Visual (AV) data that are developed and refined within this work package (Task 3.3) were considered for inclusion in the MVP, and the most relevant and mature of those offerings are now part of the demonstrator.
- From WP4, progress in various directions has provided important foundations and inputs to the MVP, namely the optimised audio capturing (Task 4.1), audio anonymisation (Task 4.2), and the Decision-Making Toolkit (T4.4).

Within WP5, there has been a close collaboration with Task 5.1 and Task 5.2, with regards to the underlying infrastructure and resource management, respectively. Additionally, there is a connection to Task 5.4 that aims to set the benchmarks to validate MVP results. This close dependency is expected to continue until the release of the final complete version.

This deliverable will be a reference to upcoming deliverables '*D5.4. MARVEL Integrated framework – initial version'*, *'D5.6. MARVEL Integrated framework – final version'*, and *'D5.7. MARVEL's framework large scale deployment'* that will describe the expanded integrated versions of the framework. The MVP demonstrator will be the basis for the first evaluation that will be part of '*D5.5. Technical evaluation and progress against benchmarks'*.

Finally, most of the work planned for 'WP6 – Real-life societal experiments in smart cities environment' will use information contained in this deliverable, but more importantly upcoming deliverable '*D6.1 - Demonstrators execution – initial version'*.

1.5 Structure of the document

The structure of this document is as follows:

- *Section 2* discusses the envisioned CI/CD processes to be established and describe which of the aimed CI/CD activities have been utilised for the development of the MVP.
- *Section 3* provides the scope and building blocks of the MVP, namely: a) a **definition of the MVP** that sets the scope and value of this release, b) the **use case selected** for demonstration and the rationale behind this choice, and c) a comprehensive **list of MARVEL components** that implement the selected use case.
- *Section 4* describes issues related to the integration of the above building blocks, namely the architecture, infrastructure, and components. Moreover, it discusses relevant deployment and testing issues that have arisen during this process.
- *Section 5* presents the MVP demonstrator, specifically from the end-user perspective, by means of Use Case implementation and User Interface screens.
- *Section 6* links the results presented in previous sections to the overall MARVEL goals and objectives.
- *Section 7* summarises and concludes this document

2 Continuous Integration

Before embarking on the definition and development of the MVP release, we need to define and put in place the integration process, i.e., the steps to take and tasks to complete so that any component that needs to be part of the next release, can be integrated easily and in a standardised way.

The main challenge for integration is to make sure that a large number of independent, heterogenous components can communicate effectively with each other, and together achieve a goal of greater scope than their individual functionalities offer. In addition to the interfaces that are required for this communication, infrastructure issues arise, as these components should operate in a well-defined environment.

Traditional approaches to integration advocate for a predefined list of releases to be realised in the future and a series of activities (design, development of interfaces, deployment, integration, testing, etc.) to occur during the time between the releases. Each of these phases is completed before moving to the next, and when all are completed, the integration and deployment is realised. This process resembles a traditional, waterfall approach to software development. However, this approach is not resilient to frequent changes in requirements and occurrence of unknown issues that are common to digital product and platform development.

For the integrated MARVEL framework, we follow an agile approach to integration, namely the Continuous Integration/Continuous Delivery (CI/CD) [2]. According to this approach, the delivered framework is implemented in small iterations, with the addition of small increments of services and functionalities at each iteration. This approach respects the natural, incremental way of developing complex systems while enabling stakeholders to monitor the implementation progress, give early feedback, and react promptly to potential technical or other obstacles that may arise. Finally, with continuous integration, qualitative, non-functional aspects of the developed platform are considered early on, including interoperability, scalability, accountability, transparency, responsibility, and performance, thus achieving quality assurance in system development iterations and releases. A typical agile, incremental process to software development is depicted in Figure 1.

Figure 1. Steps of an agile, iterative development process

In deliverable *'D1.3 – Architecture definition for MARVEL framework'* Section 2.5, we outlined such a process by listing the steps and tools required to fulfil MARVEL's technical challenges.

During the development of the MVP, we have initiated the definition of steps and the selection and configuration of tools that we will use throughout the course of the project. In the list below, we describe what has been already done during the implementation of the MVP, with reference to the guidelines provided in deliverable D1.3, Section 2.5. We also discuss actions to be taken after the MVP, towards the complete integrated versions.

- **Technical project organisation**: A detailed roadmap should be in place before the integration efforts are initiated. At integration initiation, the roadmap should contain scenarios and use cases to be implemented, a list of components and their specifications/requirements, required infrastructure needed, and planning of upcoming releases. Initial execution plans for MARVEL use cases/pilots are provided in D1.2.

  During the MVP: We have created and used such a roadmap with a comprehensive list of tasks to be carried out, with the respective timeline, milestones and expected deadline. Progress was tracked within the context of scheduled, frequent MVP-focused meetings, and details for the tasks were shared with live online documents or email exchanges.

  After the MVP: We will define a similar roadmap for the upcoming first complete release (M18). We will replace the documents and email exchanges of the MVP with an issue ticketing system (e.g., Redmine[1]) that is more efficient in tracking progress and exchanging information for ongoing and future tasks.

- **Source version control system**: During the execution of integration, all open-source components under development should be stored in a version control system (VCS, e.g., git[2]/GitLab[3]). Furthermore, participating components should ideally be developed and delivered as containerised microservices, in all cases that it is feasible, to further facilitate automation. In case a component cannot meet this requirement, integration and deployment will be realised in a manual or semi-automatic way. Each component should a) expose an interface to the other components, b) be self-sufficient and have all needed external libraries and other dependencies already installed in the container, and c) provide detailed documentation of at least its exposed interface, input/output data format, user manual (if applicable), as well as build, deployment, and execution instructions.

  During the MVP: We have put in place and use a GitLab repository[4] where all components that participate in the MVP scenarios are uploaded. All components fulfil requirements a) and b) mentioned above, as they provide interfaces to other components and are shipped using docker.

  After the MVP: We plan to provide detailed documentation for all MARVEL components that are integrated into the framework and make this documentation available to all MARVEL partners. Additionally, we will provide a standardised

[1] https://www.redmine.org/

[2] https://git-scm.com/

[3] https://gitlab.com/

[4] http://registry.marvel-platform.eu/

format for data modelling and data exchange, that all connected components will be required to implement. This way, MARVEL components will be easily integrated, while the MARVEL framework will be easily extended with future technologies and interoperable with external systems that follow the same standards.

- **Continuous Integration/Continuous Delivery**: At each iteration, a functional subset of the platform is going to be delivered for testing and demonstration purposes. As integration advances, the delivered platform will contain an increasing number of components and services, gradually reaching the full version of MARVEL. In this stage, automated tools (e.g., Jenkins[5]) will be used to facilitate the deployment processes, from retrieving the component executables, for example in the form of a docker image if applicable, to orchestrating the execution of multiple components on the specified infrastructure.

  During the MVP: No automation tools were required for the MVP, as there are only a few components that comprise the framework. Additionally, the infrastructure was configured from scratch, making connection, access, and utilisation processes available to the technology providers during the development of the MVP. Therefore, we decided it is more efficient to carry out all integration activities in a manual or semi-automated way.

  After the MVP: As the MARVEL framework will grow in scope, participating components and scale of the underlying infrastructure, adoption of automated tools for CI/CD will be beneficial.

- **Quality Assurance**: It is important to guarantee that each delivered increment meets high standards of quality both in terms of design and code implementation, as well as in terms of execution reliability, performance, and interoperability with other components. To that end, we can use automated tools (e.g., Sonarqube[6]) for code quality, test coverage, etc., in conjunction with the realisation of functional, integration, and acceptance testing efforts.

  During the MVP: Such quality assurance tools were only used partially or independently for some components of the MVP. This approach was sufficient for a small-scale, proof-of-concept demonstrator.

  After the MVP: As the MARVEL framework will grow, such tools will be incorporated in the CI/CD pipeline.

- **Bug tracking**: During the development and testing of the MARVEL platform, any bug or other system instability should be promptly recorded and made available to developers for fixing. This can be achieved by using a backlog tool that is part of the project organisation step.

  During the MVP: As mentioned above, no bug tracking or ticket issuing tool was used for the development of the MVP.

  After the MVP: We consider using such tools for the upcoming release, in order to keep track of the rising number of technical tasks to be realised.

[5] https://www.jenkins.io/

[6] https://www.sonarqube.org/

3 MVP Specification

In this section, we set the scope and goals of the MVP by providing a definition of what an MVP is, a description of the selected use case, and a comprehensive list of components that implement the selected use case.

3.1 Definition of the MVP

A Minimum Viable Product (MVP) is a concept introduced in the context of the Lean Startup methodology [1] that tackles common issues that appear when developing a new product. An MVP is defined as "that version of a new product which allows a team to collect the maximum amount of validated learning about customers with the least effort" [1]. This validated learning comes from the early release of a minimum product that shows both the potential benefits and shortcomings or unforeseen implications of that product.

A key premise behind the idea of MVP is that we produce an actual product (which may be of reduced functionality, or a service with an appearance of automation, but which is partly manual behind the scenes) that we can show to stakeholders and observe their actual reaction to the product or offered services. Essentially, the MVP serves as a proof-of-concept that can reveal early on signs of both potential value and obstacles to anticipate in the future.

The primary benefit of an MVP is that the owner of the product can gain an understanding of the interests and opinions of stakeholders without fully developing the product. The sooner this information is available, the less effort and expense will be spent on a product that may be misdirected.

As the term MVP is gaining popularity, some misconceptions about this term may arise. Often product developers focus on the "minimum" part of the acronym, defining a subset of the envisioned functionality for the product. This approach may lead to a subset of the final product that may not reveal anything meaningful about its value and the perception of the stakeholders towards this product. The term "viable" is also important in the sense that a reduced, but complete, end-to-end service should be offered to the end-users. Of equal importance is the interpretation of the reactions and the overall evaluation of an MVP. Measures and benchmarks should be defined beforehand so that owners will know what attributes to measure and how to draw conclusions from the demonstration of an MVP.

In MARVEL, we use the MVP as the core piece of a strategy of experimentation. We hypothesise that the value offered by MARVEL is indeed feasible, demonstrable and satisfies a wide range of end-user needs and has a significant societal impact. During the design of the MVP, we paid equal attention to the terms "minimum" and "viable", while kickstarting the process of defining benchmarks and evaluation goals that will take place after its release *('Task 5.4 - Quantifiable progress against societal, academic and industry validated benchmarks').*

3.2 Use case selection

3.2.1 Rationale

As described in detail in deliverable *'D1.2 – MARVEL's Experimental protocol'*, there are three real-life experiments designed to be implemented and executed in pilots for three cities, Malta, Trento, Novi Sad. For the first two, four different use cases are defined to support the real-life experiments, while for the latter pilot, two use cases are defined. These use cases

cover a wide range of activities to be monitored and potential events to be detected, analysed and mitigated.

With the kick-off of *'Task 5.3 - Continuous integration towards MARVEL's framework realisation'*, we set a series of pilot-focused meetings in order to gain a better understanding and insights about the nature, setting, involved actors, events, and technical implication of each use case. These meetings combined detailed exposition of the real-world setting and needs of the pilot end-users, with a comprehensive examination of available technologies that are related and can address the end-user needs.

The selection process for the MVP use case was based on several criteria that were discussed during these meetings, including maturity of the use case and feasibility of completion within the given timeframe. The maturity of a use case was examined in terms of a) availability of infrastructure, b) availability of historical data necessary for the development components or training of AI models that had to be realised in parallel, and c) level of understanding of the details of the use case by both pilot owners and technology providers. In terms of employed technologies, after mapping potential utilisation of offered MARVEL components to the use case, we proceeded with the evaluation of readiness of each component, in terms of a) maturity/TRL, and b) estimated effort for integration with other components.

This fruitful and productive process advanced by selecting the favourite candidates and eliminating complex use cases or use cases that rely on components that need more time to be ready for integration. The result of this process was the timely selection of *use case 'GRN4 – Junction Traffic Trajectory'* of the Malta pilot. During this process, in addition to the selected use case that is required for the MVP, significant progress has been made towards the understanding and development of other use cases that will be implanted in future releases.

The selected use case offers a wide variety of inputs that may be managed and processed by the MVP, including vehicle identification, trajectories, sound events, and crowd counting that will showcase the potentials of MARVEL's AI infrastructure, while providing the end-user with an immediate view of the events and current state happening on street level. This use case is described in detail in the following section.

3.2.2 GRN Use case 4 – Junction Traffic Trajectory

Use case *GRN4 - Junction Traffic Trajectory Collection* is focused on the requirement of long-term data analytics that shed light on both the behaviour of road users (e.g., car drivers, motorcyclists, cyclists, pedestrians, etc.) and on gathering traffic statistics at road network junctions. This use case is of interest for long-term transport planning and evaluation. In particular, there is currently significant interest in studying active travel modes, such as cycling, walking, and more generally micro-mobility. Authorities in Malta are interested in, for example, finding the optimal position of pedestrian crossings, whether provisions for cyclists at complex junctions are adequate, and whether installed provisions are being used as intended.

This use case requires entity (traffic object) detection and its trajectory across a junction or road segment and descriptive statistics of network junction traffic. It, therefore, follows that entity detection and tracking models can be potentially used as a first processing stage followed by further processing to generate descriptive statistics.

For the implementation of the MVP, the trajectories of the cars are first extracted using the CATFlow object detection and tracking algorithm, followed by basic data analysis carried out on the output data and finally visualised on the UI.

The main data for the MVP will be obtained from the Mgarr Camera. This is a static camera installed over a road junction. The main reason this camera was chosen was due to its ability to observe a junction during all times of the day, thus a varied dataset can be produced, and various testing conditions can be considered. These include lighting levels such as day, night, and dusk as well as different weather conditions such as wind, sun, and rain. The camera includes an integrated microphone that delivers raw synchronised AV data. The CATFlow algorithm makes use of the video only, however, this data will also be used to develop and test the traffic sound event detection (SED) models and the audio/video anomaly detection models. Similarly, the same setup supports the experimental implementation of algorithms at the edge.

Figure 2 shows the junction chosen for the MVP as seen from the static camera installed for the MVP. Figure 1Figure 3 shows the actual Mgarr camera being used to monitor this junction.

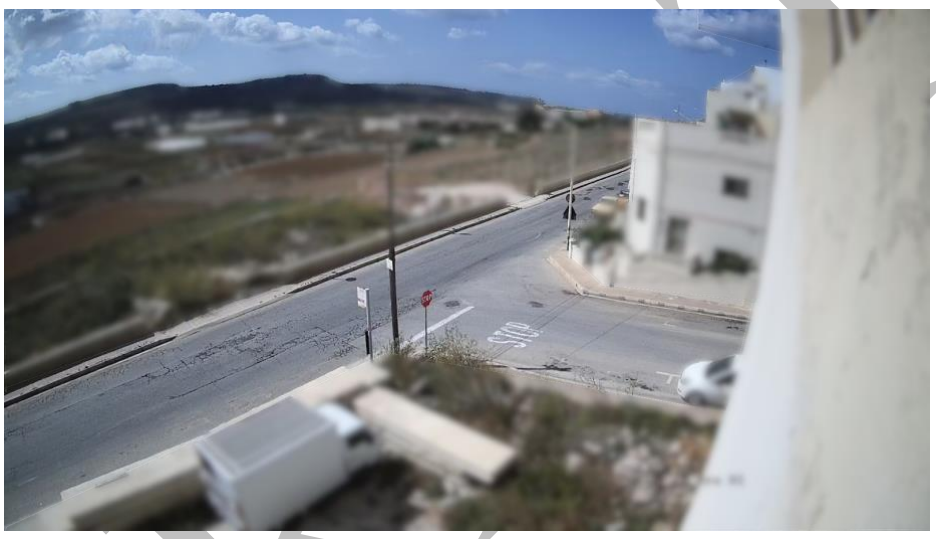

Figure 2. Junction chosen for investigation for the MVP, with certain parts blurred for privacy

Figure 3. Image of camera used to inspect junction

For this use case, we have defined two user journeys[7], i.e., operation scenarios from the point of view of the end-user. The **first user journey** for this use case concerns the collection of vehicle and bicycle trajectory data. The trajectory data is of interest since it can be used to model what could be considered normal for the different types of vehicles. This data can be either used by traffic engineers to make data-driven decisions on the trends observed at a particular road or junction or else it could be used to search for instances of anomalous events. Figure 4 and Figure 5 show an example of how vehicle trajectories change due to a lightweight vehicle stopping in the middle of the road (or at the roadside) for unloading/loading cargo. These events push cyclists and vehicles out and can be the cause of accidents. A traffic engineer would be interested in knowing whether this is a one-off event or a recurring one. If, for example, lightweight vehicles are often the cause of obstructions at this particular junction, a dedicated loading and unloading bay could be set up at the location of interest for the convenience of all road users.

[7] In this document, we use the term "User Journey" to refer to operation scenarios from the point of view of the end-user and the term "Scenario" to refer to the entire end-to-end functionalities offered for demonstration in the context of the MVP.

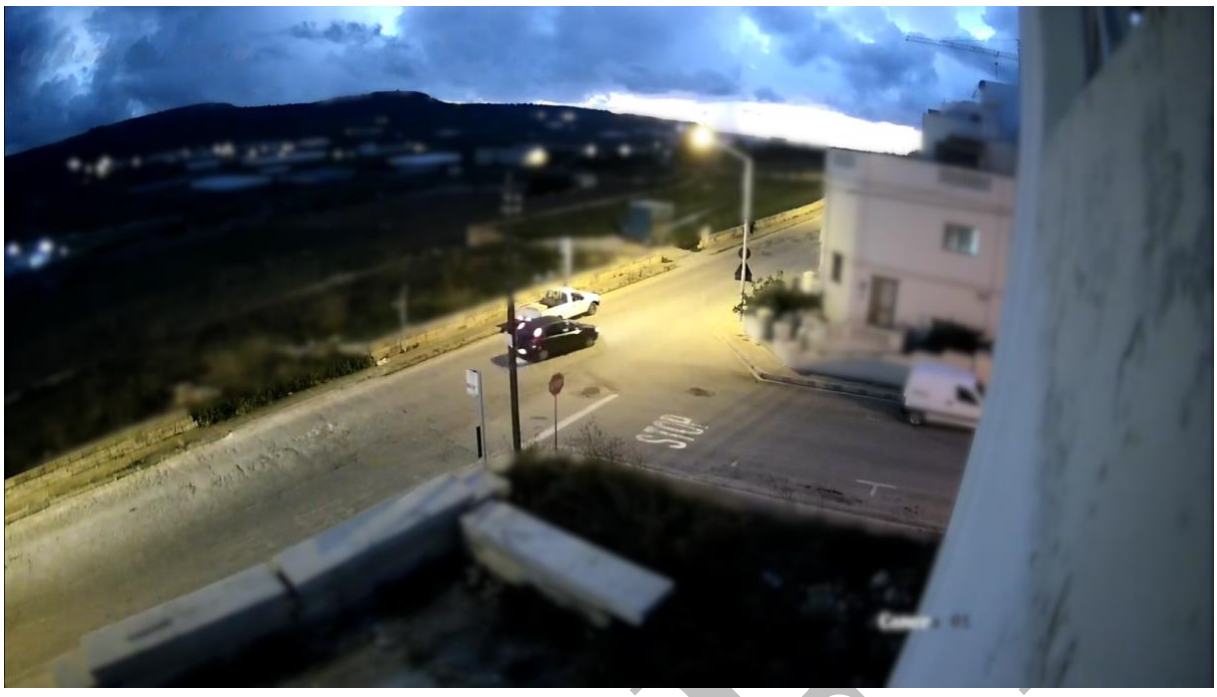

Figure 4. Still of truck causing an obstruction

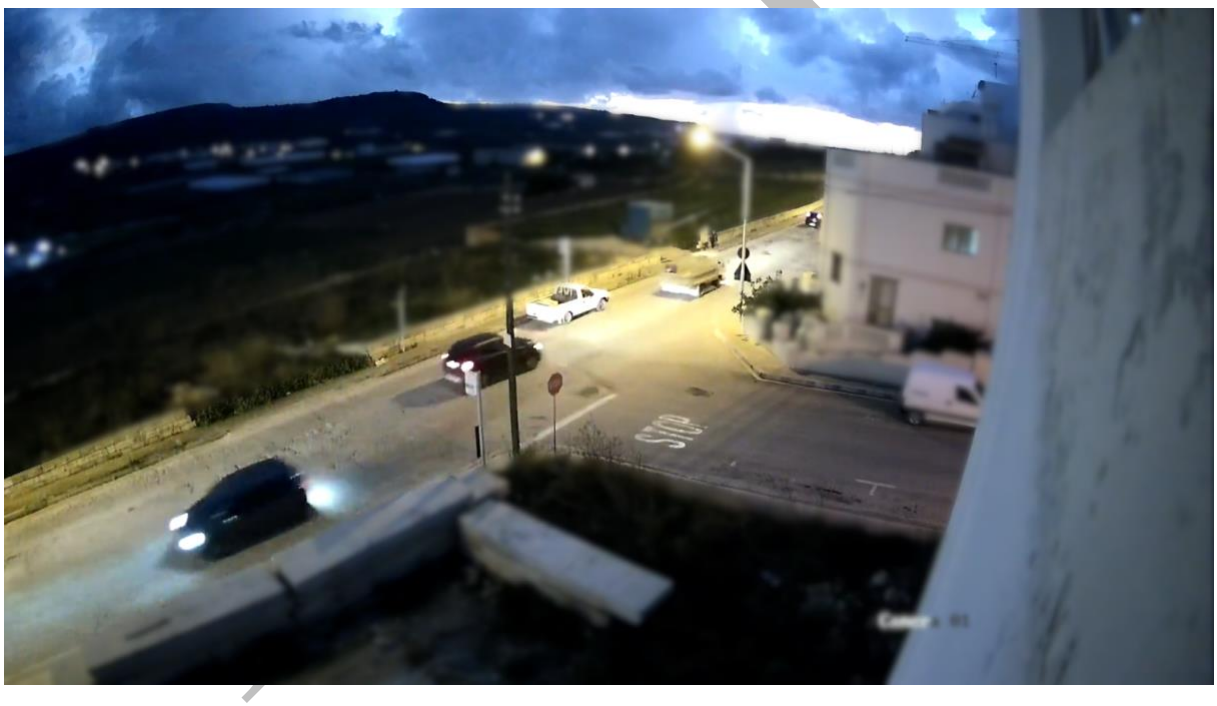

Figure 5. Still of truck still causing an obstruction and causing a change in trajectories for other cars

Other events that could be of interest for traffic engineers are manoeuvres such as three-point turns as shown in the stills below. In roads where cars frequently pass at high speeds, such manoeuvres could be dangerous and lead to accidents. This example is shown in Figure 6, Figure 7, and Figure 8.

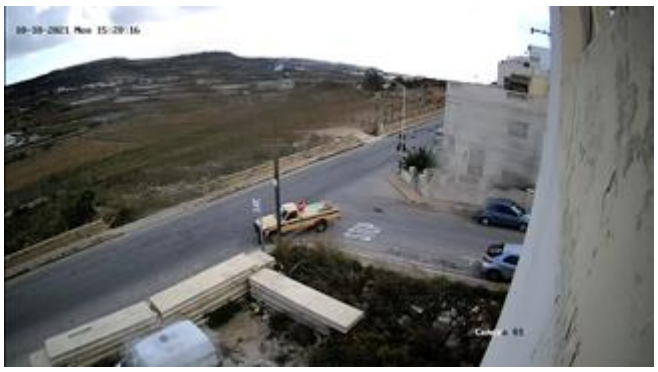

Figure 6. Three-point turn Step 1

Figure 7. Three-point turn Step 2

Figure 8. Three-point turn Step 3

The **second user journey** concerns the collection and visualisation of traffic statistics and involves mainly vehicle counting per type or class. The main purpose here for the user journey is to get data on the most frequent vehicles on the road, and thus the traffic engineer would have the ability to plan changes in the infrastructure based on the counts. The selection of this static camera placement is ideal for this user journey since it is frequented by various types of vehicles, whose distribution changes from weekdays to weekends.

Figure 9 and Figure 10 show the different types of vehicles expected at the Mgarr Camera junction.

Figure 9. Car and lightweight vehicle example

Figure 10. Bus example

3.3 Components

In this section, we provide technical information about the components that take part in one or more use case scenarios for the MVP. For each component, we provide a description, the technologies employed, and the role of that component in the MVP.

3.3.1 CATFlow

Overview

CATFlow is a system developed by GRN to analyse video footage of road traffic (possibly in real-time), to determine how road users use the given infrastructure. At the heart of CATFlow is an object detection and tracking algorithm, which detects and tracks the movements of pedestrians, cars, buses, light-goods-vehicles, heavy-goods-vehicles, motorcycles, and bicycles across the camera view. The user has the option to specify entry and exit points on carriageway lanes and these are used to compute the traffic object movement, trajectory and

estimated speed. This information is encoded in JSON format, which is then sent to the storage device ready for consumption. At the time of writing, the pedestrian data is not being collected but will be added in the future. The CATFlow output data (i.e., the JSON files or messages) does not contain any personal information. The system is currently at TRL 6 level.

Figure 11 shows an example of the detection and tracking features of the CATFlow algorithm. Each vehicle is being identified with a bounding box. The algorithm is able to distinguish between the different kinds of vehicles.

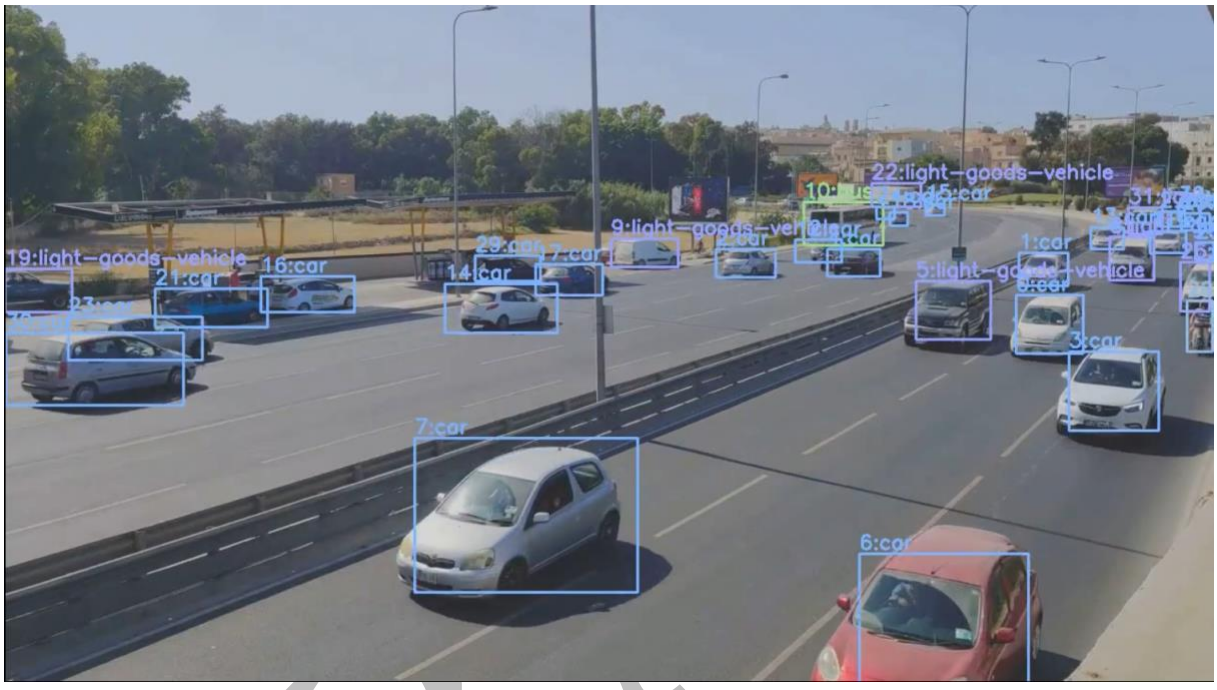

Figure 11. CATFlow algorithm example

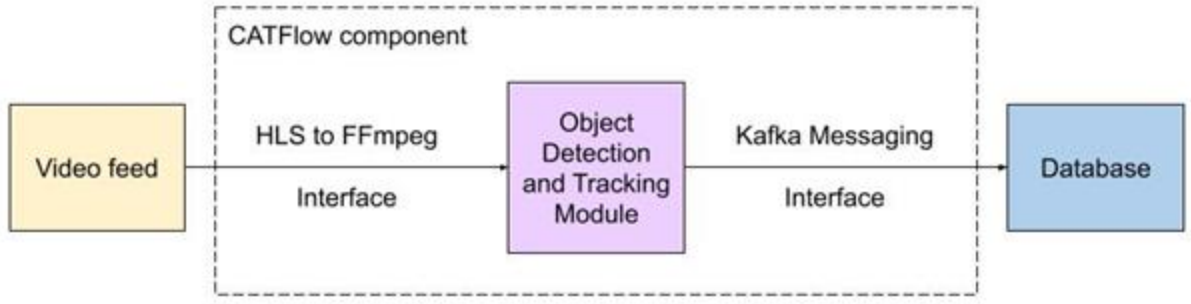

Figure 12. Dataflow to the CATFlow component

Technologies

Figure 12 depicts the dataflow in CATFlow and the interface with other components. The input is a camera stream. At the time of writing, the HLS streaming protocol is used, and then the frames are fetched using FFMPEG. Each frame is sequentially fed first into an object detector and then into a multi-object tracker in order to track entities across the camera view. Internally, it uses entry and exit lines or boundaries. Having both an entry and an exit line,

and knowing the distance between these, the vehicle speed can be estimated. Pedestrians on the other hand are tracked across the frame since pedestrians may follow a random path. No personal data is computed. Typically, the asset executes at the fog layer but can be modified to execute at the edge.

Since there can be many instances of CATFlow, Kafka is used as a message broker. The information is then stored on an OLAP database, ready to be consumed by an API to create reports or visualisations.

Role in the MVP

The camera and the CATFlow algorithm will be used as part of the assets to implement the chosen use case for the MVP, which is GRN's *Use Case 4: Junction Traffic Trajectory Collection*. The camera is for the collection of raw AV data, whilst the CATFlow algorithm will be used to extract the vehicle/object trajectories as well as traffic count data.

Use case GRN4 aims at collecting and storing the data on vehicles' trajectories. The output from the CATFlow algorithm readily yields this information on trajectories and in addition, the CATFlow output inherently includes traffic counts and estimated speed data.

3.3.2 DatAna

Overview

DatAna is an Apache NiFi[8]-based tool to manage data flows from heterogenous data sources. NiFi provides features for data collection and processing following a data-flow paradigm with very little programming effort.

NiFi's base unit of work is called a FlowFile, which provides an encapsulation of the data to be processed with some extra attributes that define that data (like filename, last update time, etc.). The NiFi FlowFiles (processing unit) can be routed and modified using specific "Processors". NiFi provides a huge number of predefined processors and extensibility to add new ones. The FlowFile passes from one processor to the next by using specific "Connections" that allows routing and branches of the overall data flow. One of the main advantages of NiFi, besides the scalability and extensibility, is the intuitive user interface that allows users to develop their data flows with little or none programming effort (although complex flows usually require some administration, monitoring, configuration, and fine-tuning).

NiFi and MiNiFi[9] (a subset of the NiFi bundle with low processing footprint for edge computing) can run independently. This means that MiNiFi agents can be deployed in devices and connection with a "master" NiFi in the fog or cloud layers can be achieved via the NiFi Site-to-Site (S2S) communication protocol[10] enabling the creation of topologies such as the ones shown in Figure 13.

[8] https://nifi.apache.org/

[9] https://nifi.apache.org/minifi/index.html

[10] https://nifi.apache.org/docs/nifi-docs/html/overview.html#high-level-overview-of-key-nifi-features

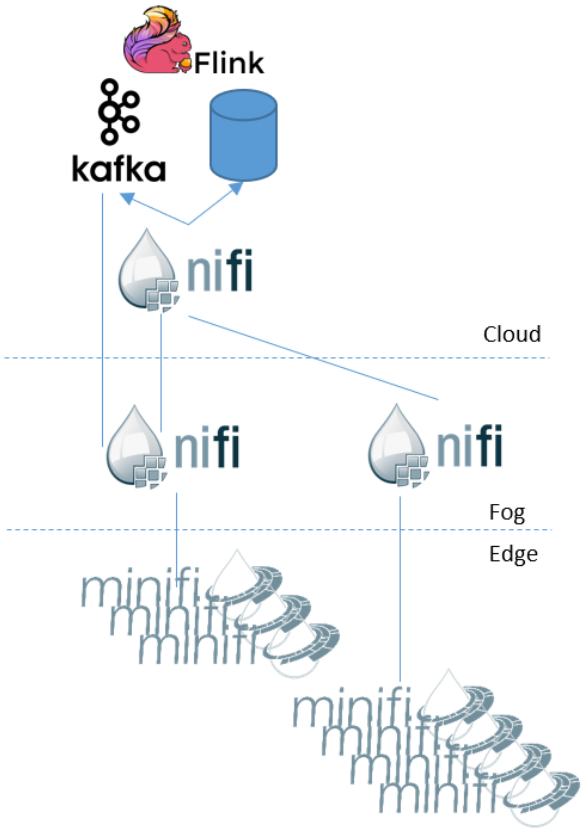

Figure 13. Apache NiFi example of topologies

The combination with other Apache NiFi subprojects, such as MiNiFi for edge devices, and the easy integration with data sources and data sinks, such as Apache Kafka, makes this tool a good candidate to glue different data processes within MARVEL.

MiNiFi agents can be installed at edge devices with some computational power (i.e., at a Raspberry Pi, NVIDIA Jetson or similar) to perform initial steps of data ingestion and processing, and then connect with other elements of the MARVEL framework, such as a NiFi or a cluster of NiFi in the fog or cloud layers. This offers the possibility of creating MiNiFi to NiFi topologies more or less complex depending on the usage scenarios and moving dataflows from one layer to the next. Figure 14 depicts a potential scenario within the MARVEL GRN pilot where DatAna is placed to handle non-AV data between the data producers (GRN) and consumers (DFB). This figure is just an example, as both DFB and DatAna could handle the data flows in different ways and different options will be explored within the project lifetime.

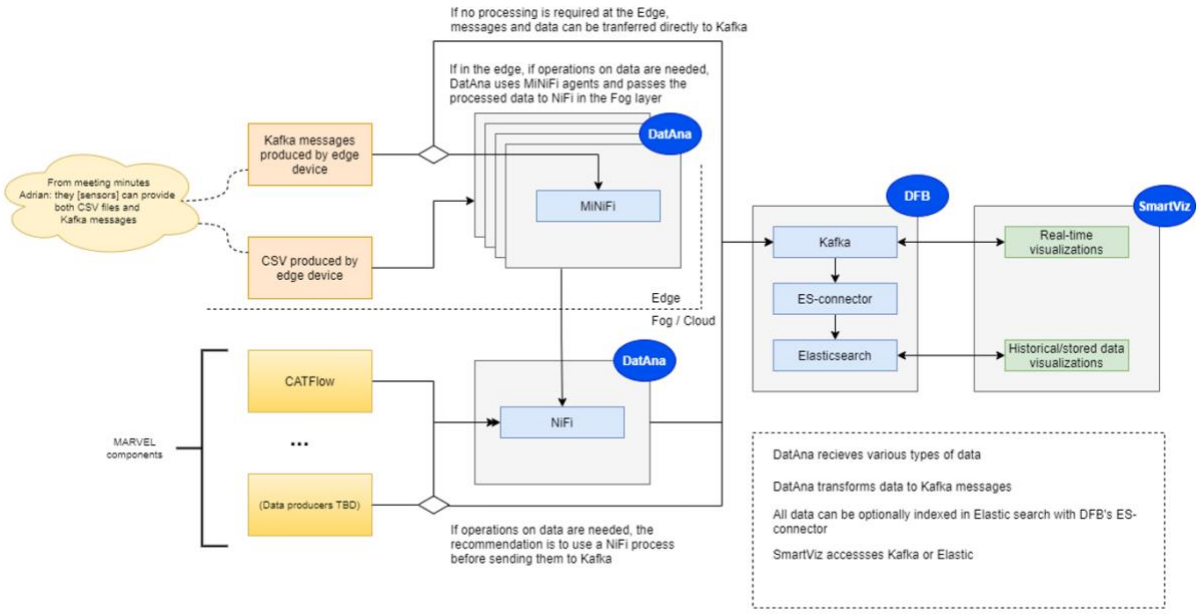

Figure 14. Example of DatAna in MARVEL

Technologies

As explained above, the main technology behind DatAna is Apache NiFi and some of its subprojects such as MiNiFi. NiFi comes with many off-the-shelf processors for data ingestion, processing, transformation and placement of data, including interfaces with Kafka, MQTT, several databases (Mongo, HBase, Cassandra, ElasticSearch, etc.), file systems and data lakes (HDFS, S3, Azure, etc.), among others. New processors can be developed and deployed.

DatAna is the new name of the tool given in MARVEL to the work done by ATOS with Apache NiFi. Besides the usage of the NiFi and MiNiFi technologies in MARVEL, the overall idea of the tool is to offer a management layer for controlling MiNiFi agents as well as helping on the right configuration of NiFi according to the best practices and needs of specific use cases. The tool in previous releases has been used in other EU-funded projects (e.g., the H2020 QROWD project) in smart cities scenarios. The current TRL is 4, and we expect to complement it within MARVEL with the new features and demonstrate its usage in a relevant environment in the MARVEL pilots to reach at least a TRL 6.

DatAna, as a DMP entry point, will interface with the data coming from the different pilot scenarios as well as with DFB via dedicated Kafka topics. From the deployment perspective, for the MVP DatAna is a simple NiFi service located in the PSNC cloud and managed by Karvdash. In future iterations this deployment will be more complex, involving MiNiFi agents at the edge and the definition of MiNiFi to NiFi topologies.

Role in the MVP

DatAna will contribute to 2 of the scenarios from the MVP as the initial entry point of the non-AV data to the DMP tools.

- Scenario 1 (identification of vehicles and trajectories): DatAna will interface with the output of CATFlow via a Kafka topic located in Azure managed by GRN. DatAna will perform a data flow to ingest this data, make some transformations, and move the results to a Kafka topic provided by the DFB inside the Karvdash environment.

- Scenario 2 (sound events and crowd counting): DatAna will implement the data flow to ingest the outputs (alerts and anomalies) from the analytical components, specifically the Sound Event Detection (SED) and Audio Visual Crowd Counting (AVCC) components, and finally pass these results to the DFB.

3.3.3 Data Fusion Bus

The Data Fusion Bus (DFB) is a customisable component that implements a trustworthy way of transferring large volumes of heterogeneous data between several connected components and the permanent storage. It comprises a collection of dockerised, open-source components which allow easy deployment and configuration as needed.

DFB's architectural design addresses several challenges that are raised by both the large volume and the heterogeneous nature of data from different sources, taking into consideration the needs and restrictions of the employed components. The main addressed challenges include:

- seamless aggregation of data with different structures or formats;
- a cluttering threat to the components due to the quantity of the input data;
- access of data through a common, safe, accessible interface.

Inherent to DFBs design is the efficient handling of the enormous volume of the data that need storage and manipulation, as well as mechanisms to remediate potential bottlenecks, lag, or high demand on network traffic. These design decisions enable horizontal scalability while providing a solution that is cloud-native with stateless components capable of being deployed with flexibility. DFB follows the middleware approach by aligning data streams for time and granularity and creating a user interface that serves as the interface of the platform, customised to aggregate multiple streams, thereby allowing seamless service of data to the network analysis and visualisation.

The key capabilities of DFB are:

- Data aggregation from heterogeneous data sources and data stores.
- Real-time analytics, offering ready-to-use ML algorithms for classification, clustering, regression, and anomaly detection.
- An extendable and highly customisable User Interface for Data Analytics, manipulation, and filtering, as well as functionality for managing the platform.
- Web Services for exploiting the platform outputs for Decision Support.
- Applications for Smart Production, Digitisation, and IoT, among others.

The key modules of DFB are:

a. *Apache Kafka*, an open-source framework for stream processing.
b. *Elasticsearch*, a distributed, multitenant-capable, full-text search engine.
c. *DFB Core & UI*, implementation of a REST API and a client GUI, respectively, for management and monitoring of the DFB components.
d. *Keycloak*, an open-source software product that provides single sign-on to applications and services.

Figure 15 depicts DFB's overall inner architecture and its relation to other components of the MARVEL platform. This figure shows DFB's main modules mentioned above, as well as its interfacing with edge devices that may produce non-binary data and other fog/cloud components that provide their processed data into DFB.

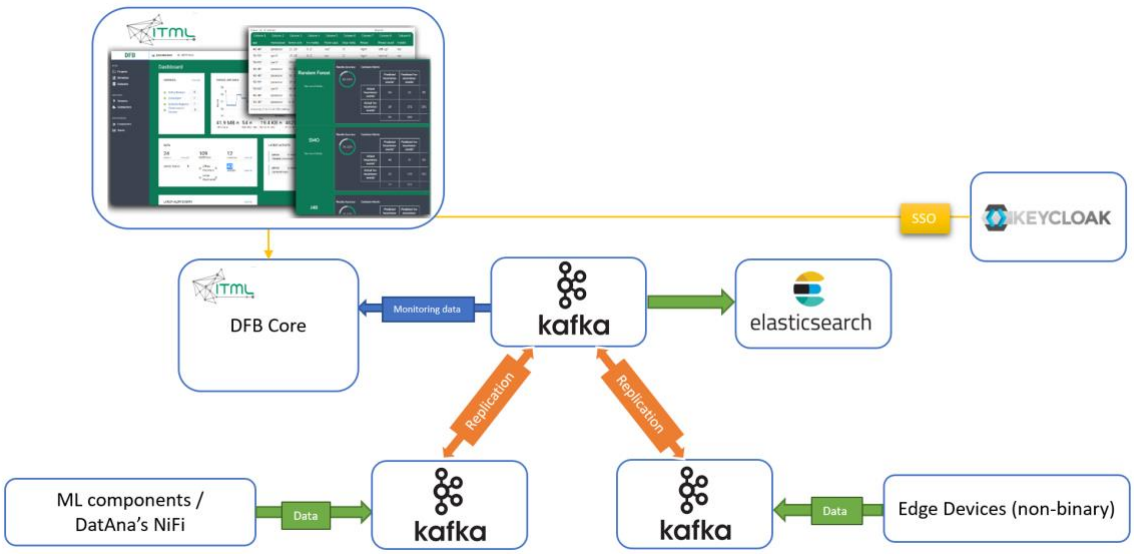

Figure 15. DFB overall architecture

The main inbound interface for DFB is Kafka's messaging system that is based on the publish-subscribe pattern. More specifically, any MARVEL data-producing component, including real-time event detection and data analytics components, connects to DFB and publishes any relevant data to a specific, predefined topic.

Regarding outbound interfaces, in the general case, any data-consuming component can subscribe to a topic and receive instant updates on published data. Within the context of MARVEL, data collected from Kafka brokers is subsequently passed onto an Elasticsearch Logstash Kibana (ELK) stack for storage and further processing and visualisation. Although DFB offers its own graphical UI for visualisation of aggregated data, collected streamed and stored data can be made available to MARVEL's visualisation components (e.g., SmartViz).

Finally, DFB offers a complete, standalone Single Sign-on module based on Keycloak open-source product to ensure authenticated and authorised access to fused data.

As DFB is typically deployed on the cloud, it does not offer any edge processing options. It is designed and optimised for handling non-binary data that is streamed to the Kafka interfaces in real-time. For the MVP, DFB will not directly process or handle AV data; instead, any results of AV processing from ML components can be made immediately available for real-time analytics or stored for later filtering, processing, searching, and visualisation.

Role in the MVP

DFB will contribute to the 3 use cases from the MVP as the engine that fuses and indexes large volumes of heterogenous non-AV data. It serves as an interface point for transferring non-AV data collected by the DMP platform, especially DatAna, to the SmartViz's UI, as well as with connections to the MARVEL Data Corpus metadata.

In all scenarios, the input to DFB is non-AV data collected and processed by DatAna. The latter component acts as a producer to DFB's Kafka entry point, by subscribing to specific topics that are defined per scenario. The data streamed through Kafka are indexed in DFB's Elasticsearch, using DFB's ES-connector.

The data streamed through or indexed at DFB are readily available for any client component, most importantly SmartViz. This user-facing component can access data in two ways: a) by subscribing to the corresponding topic of DFB's Kafka, for real-time visualisations, or b) by querying DFB's Elasticsearch API, for visualisations of historical data.

3.3.4 GRNEdge

Overview

For the purpose of the MVP (for which AV computation will not take place at the edge) the GRNEdge component is reduced to an off-the-shelf audio-video bullet camera. The camera is connected to a modem to stream AV data directly to the Fog server. For the time being, this component has no ability to carry out computations at the edge.

This GRNEdge device is located on the road that leads into and beyond a rural and largely residential but also touristic town with a clear view of the two-way road and a side street. In addition, the camera targets two bus stops on opposite sides of the road, ensuring the collection of data that includes pedestrian traffic. In general, the road is used by various types of vehicles such as bicycles, motorcycles, cars, heavyweight and lightweight vehicles, and buses.

The camera in a static location allows us to view the situation on the road during various times of the day and in different weather conditions. The constant streaming also gives us the possibility to catch anomalous events such as car crashes or illegal manoeuvres of vehicles. The TRL for this GRNEdge variant is TRL 7. Figure 16 shows the image streamed from the camera at dusk.

Figure 16. Image from camera at dusk

Technologies

The component is a Safire2MP Dome Outdoor/Indoor IP Video Camera with POE & built-in microphone. Thus, the microphone is in the same casing as the camera. The video is being compressed with H264 compression. The streaming is in real-time to the GRN server and makes use of the RTSP streaming protocol.

Role in the MVP

This component has various uses in the MVP. The first scenario where the GRNEdge device will be used is as input to the CATFlow algorithm. The GRNEdge device will stream directly to the fog server where the CATFlow algorithm processes the video stream to detect and track the vehicles.

The Mgarr camera is also being used to collect data for training and testing the algorithms which will be implemented in the MVP.

3.3.5 Karvdash

Overview

Karvdash is a service for facilitating interaction with MARVEL's E2F2C testbed, by supplying the landing page for users working on the platform, allowing them to launch services, design workflows, request resources, and specify other parameters related to execution through a user-friendly interface. Karvdash aims to make it straightforward for domain experts to interact with resources in the underlying infrastructure without having to understand lower-level tools and interfaces.

Karvdash is deployed upon a Kubernetes installation as a service and provides a web-based, graphical interface to:

- Manage services or applications that are launched from customisable templates.
- Securely provision multiple services under one externally accessible HTTPS endpoint.
- Organise container images stored in a private Docker registry.
- Perform high-level user management and isolate respective services in per-user namespaces.
- Interact with datasets that are automatically attached to service and application containers when launched.

Karvdash does not do any processing but provides the mechanisms to efficiently deploy the services included in the MARVEL data management toolkit, in all computing continuum layers that support container-based execution. To configure and start services, Karvdash uses a service templating mechanism - each service is defined with a series of variables. The user can specify values to the variables as execution parameters through the dashboard before deployment, and Karvdash will set other "internal" platform configuration values, such as private Docker registry location, external DNS name, and others.

Internally, at the Kubernetes level, each Karvdash user is matched to a unique namespace, which also hosts all of the user's services. Containers launched within the namespace are given Kubernetes service accounts which are only allowed to operate within their own namespace. This practice organises resources per user and isolates users from each other.

Karvdash is currently at TRL 3, the goal being to reach TRL 5 as part of the integration into the MARVEL platform.

Technologies

Karvdash is written in Python using the Django[11] framework. It is developed and tested on Kubernetes 1.19.x-1.21.x. Every new Karvdash release is packaged in a container image and can be installed using Helm[12], as is the typical case with software packages on top of Kubernetes.

For the purpose of the MARVEL MVP, Karvdash in installed on a VM deployed on PSNC's OpenStack offering. The VM is assigned 8 CPUs, 16 GB RAM, 512 GB of block storage, runs Ubuntu 20.04.1 LTS Linux, and Kubernetes 1.19.8. This VM is placed at the "cloud" side of the E2F2C platform.

Once deployed in the Kubernetes environment, Karvdash interfaces with the following software components (installed as Karvdash pre-requisites) and platform facilities:

- The *cert-manager*[13] certificate management controller for Kubernetes. This is used for creating certificates automatically for the Kubernetes mutation/admission webhooks that implement automatic storage mounting in user containers.
- The *ingress controller*[14], which answers to a domain name and its wildcard; in MARVEL, the ingress serves both "marvel-platform.eu" and "*.marvel-platform.eu". The ingress controller (under Karvdash control) is responsible for implementing the platform's external HTTPS endpoints, consolidating them all under the same DNS suffix, and routing each different endpoint to the appropriate running service.
- *JupyterHub*[15] for composing code in *notebooks* and *Argo Workflows*[16] for orchestrating tasks as workflow graphs. Karvdash runs side-by-side with these services, providing SSO to users. For Argo Workflows, Karvdash also configures appropriate authorisation directives, so each user will be allowed to access resources in the corresponding Karvdash-defined namespace.
- A private container registry for storing container images locally.
- Karvdash uses local storage for state-keeping and data files, via-a claim to an existing persistent volume on locally mounted block storage.

Role in the MVP

Karvdash plays a central role in all use cases included in the MVP by coordinating the execution of the data management platforms and other software components on the E2F2C testbed, as well as by mediating external accesses to any service that needs to be exposed outside the MARVEL infrastructure (both taking care of connectivity, as well as authentication issues). Moreover, Karvdash provides the implementation basis for optimised deployment of AI tasks (as per project Task 3.4).

For the DatAna and DFB data platforms, we have created Karvdash-compatible templates, so any platform user can launch the corresponding service. All containers are launched within the user's private namespace, while respective frontend's (DatAna's NiFi GUI and DFB's Kibana GUI) are exposed via Internet-accessible URLs, over HTTPS, and protected using the Karvdash authentication infrastructure. Internally, these services can access any resource in the E2F2C-private network. Persistent data is directed to Karvdash-supplied storage that can

[11] https://www.djangoproject.com
[12] https://helm.sh
[13] https://cert-manager.io
[14] https://kubernetes.github.io/ingress-nginx/
[15] https://jupyter.org/hub
[16] https://argoproj.github.io/workflows

also be accessed via the Karvdash web interface. Deployment of DatAna and DFB also required customisation and deployment of dependent ZooKeeper and Kafka services.

Karvdash also assists in the deployment of the Data Corpus, by integrating a web proxy service to allow external, secure access to the Data Corpus API services. The proxy implementation registers a new ingress endpoint at the Kubernetes side (in order to get a HTTPS-compliant URL) and is configured to require authentication using the Karvdash-registered credentials before forwarding requests to the backend.

3.3.6 Sound event detection

The sound event detection (SED) component takes as input audio signal captured at the scene and outputs sound event activity information in pre-specified time-resolution (one second). For each detected sound event, the component provides the sound class label and temporal information when the sound activity started (onset timestamp) and when it ended (offset timestamp). The timestamps will be reported in the relation to the start of the signal. The overview of this SED is shown in Figure 17.

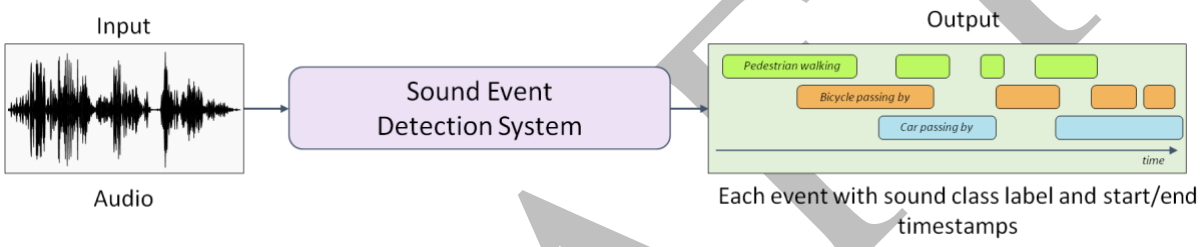

Figure 17. Illustration of the sound event detection component

The SED component implementation for the MVP detects four types of road vehicles: bus, car, motorcycle, and truck. To develop a robust sound event detection system, a large dataset captured in real conditions and manually annotated with the onset and offset timestamps for the sound events (i.e., strong annotation) is required. As the manual annotation is not feasible for such large datasets, the SED component is implemented using the transfer learning approach. This approach uses a pre-trained model as a feature extractor to extract audio embeddings. These embeddings are then used as input to learn target task-specific the SED component for the MVP task. The component uses a pre-trained PANN model (MobileNetV1) [3] that is trained with the AudioSet dataset (5000h of audio material) for a general audio tagging task. The knowledge learned by this model is then transferred to vehicle detection task by using task-specific learning examples to train a new model while using audio embeddings produced by the pre-trained PANN model as features. The learning examples are from task-specific public datasets (MAVD traffic dataset (4h audio) [4] and IDMT traffic dataset (2.5h audio) [5]) as well as recordings from the GRN (recordings from mobile setups and from fixed camera setups).

3.3.7 MEMS microphone IM69D130

The IM69D130 MEMS microphone is designed for applications where lower self-noise, wide dynamic range, low distortions and a high acoustic overload point is required. It features the specific IFAG Dual Backplate MEMS technology. Its main features are:

- Dynamic range of 105 dB
  - Signal to noise ratio of 69 dB(A) SNR
  - <1% total harmonic distortions up to 128dBSPL

  - Acoustic overload point at 130dBSPL
- Sensitivity (± 1dB) and phase (± 2° @1kHz) matched
- Flat frequency response with low frequency roll-off at 28Hz
- Very fast analog to digital conversion speed (6µs latency @1kHz)
- Power optimised modes determined by PDM clock frequency
- PDM output
- Omnidirectional pickup pattern

In the scope of MARVEL, these features will enable accurate and clear data acquisition.

Audiohub Nano data acquisition component: This component features two microphones for audio data acquisition and transfers the PDM input from the microphones to USB output, such that the device can be connected via USB to a further processing device. Crucial features of the device are:

- Audio streaming over USB interface
- Powered through Micro USB
- 48 kHz sampling rate
- 24-bit audio data (stereo)
- Mode switch for toggling between normal mode and low power mode with four pre-defined gain configurations
- LED indication for the configured gain level in normal mode and low power mode
- Volume unit meter display with onboard LEDs

It can be easily used with Edge Devices due to the omnipresence of USB connectors.

3.3.8 Visual crowd counting

This component takes as input an image or video frame of a scene that may or may not involve crowds of people, and outputs a number representing the total number of people present in the image or video frame, which may be zero if no people are present. Optionally, this component may also output a density map indicating the density distribution of the crowd in each part of the image or video frame.

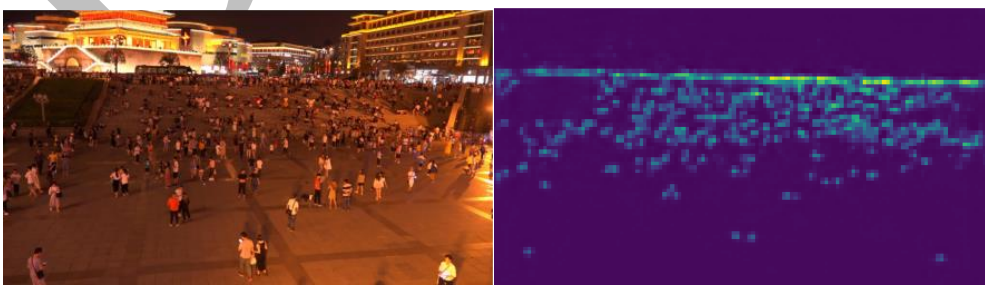

Figure 18. Sample of a scene involving crowds and its corresponding density map

Crowd counting is a regression problem which we approach using DL. Deep neural networks provide outstanding results; however, they are typically made up of many interconnected layers, which makes them computationally expensive and slow. Dynamic inference methods

alleviate this problem by skipping parts of deep neural networks, however, the downside is that the results are usually less accurate. One such dynamic inference method is early exiting, where early exit branches are added after intermediate layers of the deep neural network and provide early results. We have proposed several methodologies for decreasing the accuracy drop in these early exits, for instance, using curriculum learning, which first introduces easier examples to the deep neural network and gradually introduces more difficult ones during training; similar to how humans learn a curriculum.

Audio-visual crowd counting component

Similar to the visual crowd counting component, this component takes as input an image or video frame of a scene that may or may not involve crowds of people. Additionally, a short audio clip starting from -0.5s before the image or video frame was taken and ending +0.5s afterwards, representing the ambient audio of the scene, is also given as input. This ambient audio can help in situations where the quality of the image is low, for instance, low illumination, low resolution, capture device noise and occlusion. The output is a number representing the total number of people present in the image or video frame, which may be zero if no people are present. Optionally, this component may also output a density map indicating the density distribution of the crowd in each part of the image or video frame.

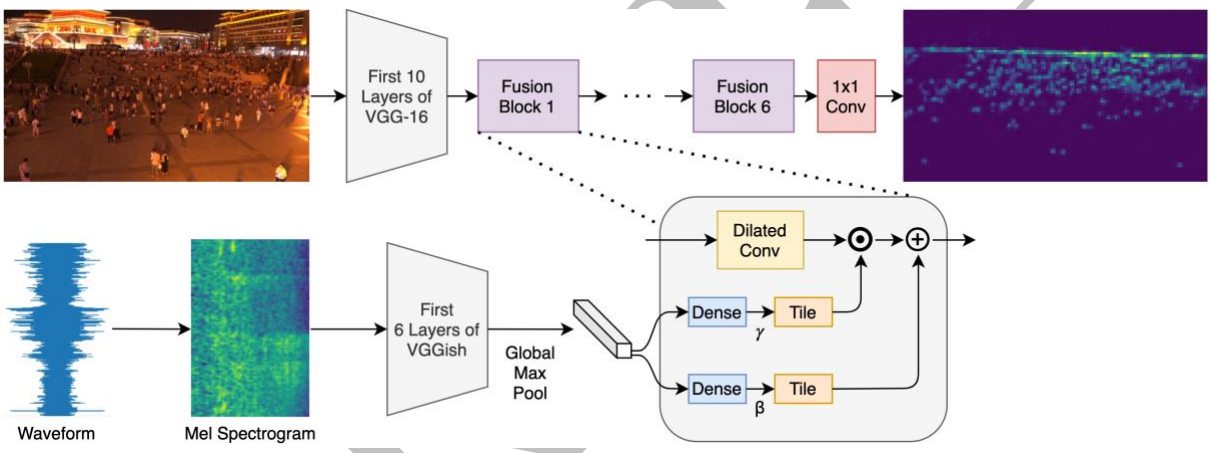

Figure 19. Schematic illustration of the audio-visual crowd counting component

As previously mentioned, crowd counting is a regression problem which we approach using DL. Deep neural networks provide outstanding results, however, they are typically made up of many interconnected layers, which makes them computationally expensive and slow. Dynamic inference methods alleviate this problem by skipping parts of deep neural networks, however, the downside is that the results are usually less accurate. One such dynamic inference method is early exiting, where early exit branches are added after intermediate layers of the deep neural network and provide early results. We have proposed several methodologies for decreasing the accuracy drop in these early exits, including a novel approach of mixing audio and visual features directly in the early exit branch using the newly introduced Vision Transformer architecture [6].

3.3.9 MARVEL Data Corpus

MARVEL Data Corpus is a service for facilitating the ingestion and retrieval of processed multimodal AV open data, obtained free of charge. Currently deployed behind Karvdash, MARVEL Data Corpus works as a central repository offering a series of REST APIs for uploading and accessing AV data.

Towards the MVP, MARVEL Data Corpus has been deployed as a fully distributed dockerised environment using two basic technologies HBase[17] and Hadoop[18]. HDFS is a File system of Hadoop designed for storing very large files running on a cluster of commodity hardware. Hadoop HDFS provides a fault-tolerant storage layer for Hadoop and its other components. It provides high throughput access to application data by providing the data access in parallel. On the other hand, HBase is a Hadoop database, a distributed, scalable, big data store. HBase provides low-latency random reads and writes on top of HDFS and it's able to handle petabytes of data.

Currently, MARVEL Data Corpus consists of:

- An HDFS namenode that regulates file access to the clients. It maintains and manages the datanode and assigns tasks to them. Namenode executes file system namespace operations like opening, closing, and renaming files and directories.
- An HDFS datanode that manages storage of data. This slave node is the actual worker node that does the tasks and serves, reads, and writes requests from the file system's clients.
- An HDFS node manager as a WebLogic Server utility that enables to manage namenode and datanode instances remotely.
- An HDFS resource manager node that is responsible for tracking the resources and scheduling applications (such as Spark[19] Jobs).
- An HDFS history server that works as a staging directory for temporary file creation while running jobs also as directory for log files of finished jobs.
- An HBase master node that coordinates the HBase Cluster and is responsible for administrative operations.
- An HBase region server that is responsible to handle a subset of the table's data.

For the MVP, MARVEL Data Corpus exposes two basic REST APIs functionalities: a REST API for ingesting data to the Corpus and a REST API for creating the respective record inside the HDFS. The ingestion procedure, regarding the first API, is performed in two parts: first, a negotiation with the HDFS for creating the necessary resource inside the Hadoop file system which returns a unique location-endpoint inside the HDFS system and second, the actual ingestion procedure to the latter endpoint. The REST API for creating records to the HBase system is based on the description of audio and video metadata. The records will also include a key-value field that will have the exact location-endpoint of the HDFS storage position of the data.

3.3.10 SmartViz

Overview

SmartViz is a data visualisation toolkit that will constitute the UI of the decision-making toolkit. It provides the means to visualise several incoming detected events deriving from the analysis of data coming through the Edge layer. It consists of a set of visualisation tools developed to allow exploratory analysis of the incoming data by using interactive and relative presentations to the nature of the data. SmartViz enables the end-users to interact with and

[17] https://hbase.apache.org/

[18] https://hadoop.apache.org/

[19] https://spark.apache.org/

gain a solid understanding of data, to drill into more detailed information, to discover patterns and correlations of data items and to lead them in making data-driven decisions. SmartViz functionalities will include advanced visualisations of detected events, demonstration of the related statistical data via different visualisation schemas and widgets, data filtering options and flexible interfaces.

Using its aforementioned capabilities, SmartViz aims to help users greatly decrease the time needed to gain a solid situational awareness. Therefore, decision-makers will be empowered to discover patterns, behaviours, and correlations of data items via a visual data exploration process that will be supported and enhanced by knowledge gained about the available information by the rest of the MARVEL subsystems.

The data visualisation toolkit will be able to consume a variety of data, coming through different workflows and pipelines inside the MARVEL infrastructure. Such data could have different velocity, format, type and can be offered via live or batch streams. The purpose of the SmartViz toolkit is to be able to handle and support all the above and in parallel to satisfy the end-user needs and facilitate decision-making.

SmartViz's current TRL is 4 and is expected to reach TRL 6 at the end of the MARVEL project.

Technologies

The SmartViz data visualisation toolkit is served as a web application directly accessible by end-users and it is written in Typescript using the Angular[20] framework. It uses selected visualisation libraries written in Typescript and JavaScript that contribute to the pool of visualisation widgets.

This frontend part of the toolkit depends on a middleware, that transforms information into internal data representations, which can therefore feed the visualisations. This web server that acts as the middleware between the SmartViz and the other data distribution components of the MARVEL subsystems, also hosts configuration options, managing parameters like user authentication and authorisation, dashboard sharing options, etc. It is written in Node.js[21] with Express.js[22] as its web application framework and it uses the architectural style of REST. SmartViz is capable of connecting with multiple data sources and then use its internal data API and configuration options to produce predefined as well as user-defined visualisation dashboards. Existing adapters can already support RESTful APIs and real-time data feeds (i.e., Kafka topics) through the web server described. The deployment and integration of SmartViz in MARVEL will be done in Docker environment since our services are offered as Docker images along with the corresponding configuration files (i.e., in yaml format).

Role in the MVP

SmartViz is the final component in the tail of the MPV's pipeline and its purpose is to give meaningful insights and interaction capabilities to the end-users and thus, to facilitate urban planning.

SmartViz plays a central role in two of the use case scenarios selected for the MVP. It will carry out the visualisation needs of the undertaken use case GRN4: Junction Traffic

[20] https://angular.io/

[21] https://nodejs.org/

[22] https://expressjs.com/

Trajectory, along with the specified user stories that will be described in Section 5. The pool of visualisation includes widgets that vary from simple charts and graphs to complex timeline representations, geospatial depictions, and real-time rendering of data. For the MVP, we will select several widgets from this pool that will make sense for the user needs, and for the respective data that will be utilised. Geospatial depictions and statistical analysis through graphs, charts, and tables with information regarding the detected events are some of the main widgets that will be exploited for the scope of the MVP. Regarding the middleware that is described above, we will use all the functionalities it offers, and it will support both real-time data streams and historical data.

The DMT will constitute and facilitate the interactions with MARVEL end-users. It will serve as the key interface of all processes performed in MARVEL delivering all the insights produced to the users to consume and decide the next steps. SmartViz is the component that facilitates the visualisations depicted in the DMT through pre-configured widgets according to the preferences of the users and co-designed with the MARVEL partners and the GRN pilot for the MVP of the project. For the scope of the MVP, SmartViz connects by its internal data API and configuration with RESTful APIs and real-time data feeds (i.e., Kafka topics), to produce a user-predefined visualisation dashboard. However, more data adapters can be plugged in to support different data sources for including other pilots and use cases in the upcoming versions of the DMT. The output of the adapters is handled by a middleware, that transforms information into internal data representations, which therefore feeds the visualisations. The frontend part of the tool is served as a web application directly accessible by end-users. Data are displayed using a pool of visualisations that include simple charts, graphs, and geospatial depictions in real-time rendering of data streams. Moreover, advanced filtering mechanisms and interconnected visualisations are available to allow multiple ways of data presentation and foster exploratory data analysis.

4 Integration and deployment

In this section, we present integration and deployment activities for the MVP, including the specific architectural views for the selected use case scenarios, the required infrastructure, as well as discussion of components and data integration, deployment and testing.

4.1 Architecture

We now explain the instantiation of the MARVEL conceptual architecture, presented in Figure 1 of D1.3 for the MVP use case *GRN4: Junction traffic analysis*, including also further specifications, refinements and interactions of MARVEL components relevant for the MVP. For completeness, we show the MARVEL conceptual architecture in Figure 20 below.

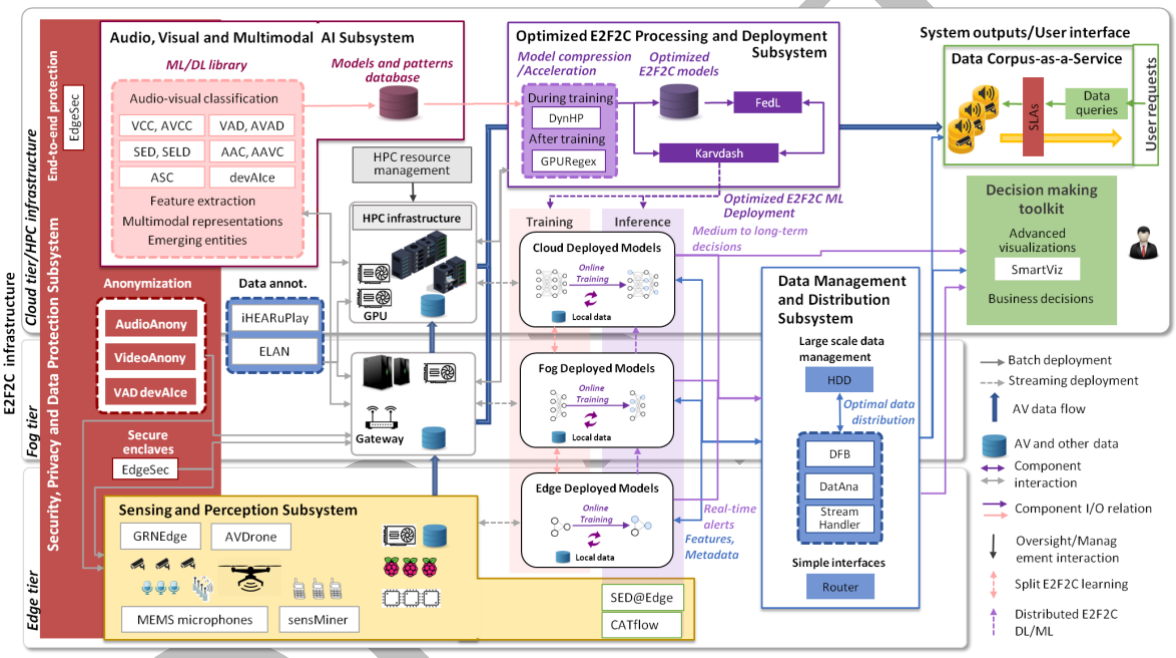

Figure 20. MARVEL conceptual architecture

In the context of the MVP demonstrations, we define three user journeys (UJ) that describe the operation of the framework as perceived by the end-user. We also use the term "Scenario" that supports and implements the defined UJ, describing not only the end-user experience but also the technical details for that implementation.

We consider three different UJs within the MVP GRN4 use case:

1. UJ1: CATFlow inference user journey: This UJ provides the end-user with information about detected vehicles and their trajectories. In particular, the end-user visualises the trajectories of the detected vehicles, as well as statistics over the usage of the junction by particular vehicle types, for example, cars, trucks, and motorcycles.
2. UJ2: Sound events and crowd counting user journey: This UJ presents the end-user with detailed advanced information, including sound events and pedestrian distribution on the area of the monitored junction. The end-user is informed about the exact time of the interesting sound event, and a heatmap of pedestrian distribution on the area of the junction.

3. UJ3: MARVEL Data Corpus user journey: This UJ lays the path of the infrastructure that supports training of AI models, by forwarding anonymised AV data to the MARVEL Data Corpus that can be monitored and queried by the Corpus admin dashboard.

The first two user journeys correspond to data/workflow for inference, while the third one corresponds to the data/workflow for MARVEL Data Corpus. In the following sections, we describe the MARVEL architecture for each of the above user journeys, specifically, the runtime and deployment view.

4.1.1 Runtime and deployment view for the GRN4-UJ1: CATFlow inference

Table 1 provides the list of components together with their main functionalities, across MARVEL functional subsystems, to be used in GRN4-UJ1. In this scenario, video streams from two cameras are collected using GRNEdge in raw form and transmitted wirelessly to the GRN fog server. CATFlow, deployed also at the GRN fog server, ingests this data in real-time and produces from the ingested videos the number of detected vehicles and traffic trajectories. These output results of CATFlow are in the JSON format and are produced periodically. The output of CATFlow is published as a Kafka topic and further ingested to DatAna using NiFi, where basic data transformation is achieved. From DatAna, the dataflow is passed to DFB and finally to SmartViz. SmartViz provides the UI to the user (in this scenario, traffic engineer), including visualisations related to the CATFlow inference results. Karvdash is orchestrating the deployment of the cloud-based components: DatAna, DFB, and SmartViz. Figure 21 provides the deployment and a simplistic runtime view of the MARVEL architecture for the GRN4-UJ1 scenario.

Table 1: MARVEL architectural components for GRN4-UJ1: CATFlow inference

MARVEL subsystem	Component information			
	Name	Owner	Functionality	Deployment layer
Sensing and perception subsystem	GRNEdge	GRN	Video streams collection and transmission (2 cameras)	Edge
	CATFlow	GRN	AI inference: vehicle count, traffic trajectories	Fog
Data management and distribution subsystem	DatAna	ATOS	Data management – NiFi	Cloud
	DFB	ITML	Data management – Kafka	Cloud
Optimised E2F2C processing and deployment subsystem	Karvdash	FORTH	Service deployment	Cloud
E2F2C infrastructure	Cloud layer	PSNC	Cloud layer for GRN4-UJ1: PSNC cloud	Cloud
	Fog layer	GRN	Fog layer for GRN4-UJ1: GRN server	Cloud
Decision-making and user interactions subsystem	SmartViz	ZELUS	UI and visualisations	Cloud

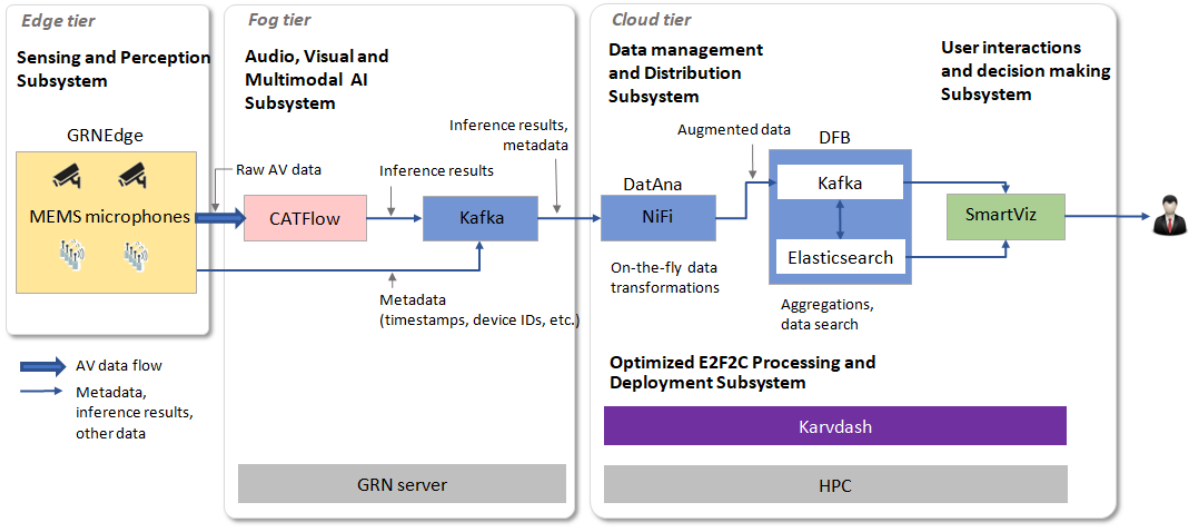

Figure 21. Scenario 1: CATFlow inference - Deployment and runtime view

4.1.2 Runtime and deployment view for the GRN4-UJ2: Sound events and crowd counting

Table 2 provides the list of components together with their main functionalities, across MARVEL functional subsystems, to be used in GRN4-UJ2. In this scenario, video streams from two cameras are collected using GRNEdge in raw form and transmitted wirelessly to the GRN fog server. Two AI components, SED and AVCC, ingest this data in real-time and produce timestamps for abnormal sound events, and pedestrian distribution over the junction. These output results of the algorithms are in the JSON format and are produced periodically. The outputs of SED and AVCC are published as Kafka topics and further ingested to DatAna using NiFi, where basic data transformation is achieved. From DatAna, the dataflow is passed to DFB and finally to SmartViz. SmartViz provides the UI to the user (in this scenario, traffic engineer), including visualisations related to the inference results. Karvdash is orchestrating the deployment of the cloud-based components: DatAna, DFB and SmartViz. Figure 22 provides the deployment and a simplistic runtime view of the MARVEL architecture for the GRN4-UJ2 scenario.

Table 2: MARVEL architectural components for GRN4-UJ2: Sound events and crowd counting

MARVEL subsystem	Component information		
	Name	Owner	Functionality
Sensing and perception subsystem	GRNEdge	GRN	AV data collection and transmission
Multimodal AI	SED	AU	Sound Event Detection
	AVCC	TAU	Audio Visual Crowd Counting
Data management and distribution subsystem	DatAna	ATOS	Data management – NiFi
	DFB	ITML	Data management – Kafka
Optimised E2F2C processing and	Karvdash	FORTH	Service deployment

deployment subsystem			
E2F2C infrastructure	Cloud layer	PSNC	Cloud layer for GRN4-UJ2
	Fog layer	GRN	Fog layer for GRN4-UJ2
Decision-making and user interactions subsystem	SmartViz	ZELUS	UI and visualisations

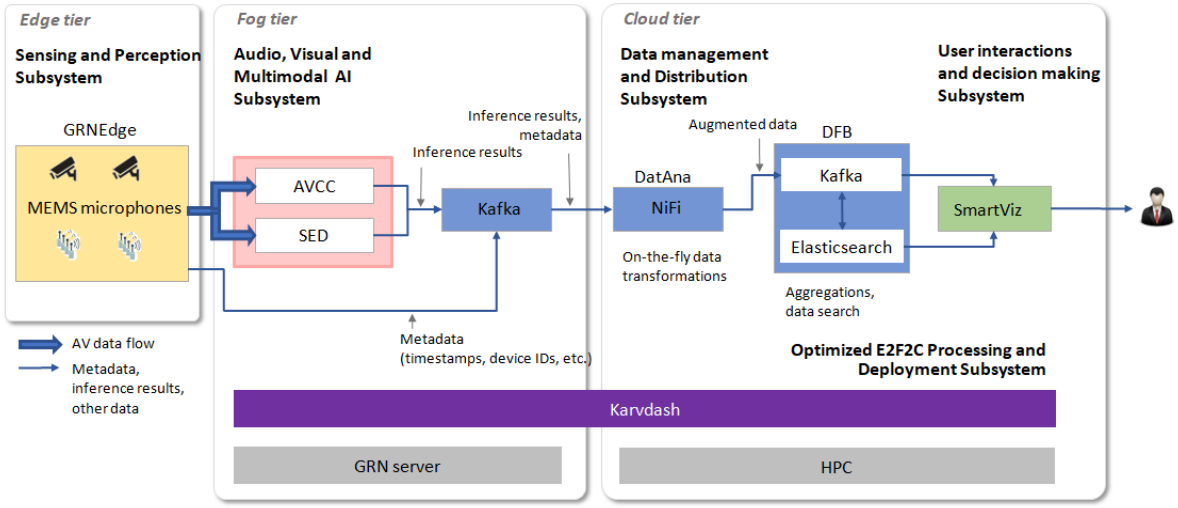

Figure 22. Scenario 2: Sound events and crowd counting – Deployment and runtime view

4.1.3 Runtime and deployment view for the GRN4-UJ3: MARVEL Data Corpus

Table 3 provides the list of components together with their main functionalities, across MARVEL functional subsystems, to be used in GRN4-UJ3. In this scenario, video streams from two cameras are collected using GRNEdge in raw form and transmitted wirelessly to the GRN fog server. These data are passed through an anonymisation component. The anonymised AV data, along with relevant metadata and annotations are processed by a script that makes sure that all data are transferred and stored in the Data Corpus, following a specific data model, defined for the needs of MARVEL. The current version of this data model provided in Appendix A, and is expected to evolve as the Data Corpus is further developed. Karvdash is orchestrating the deployment of the cloud-based components. Figure 23 provides the deployment and a simplistic runtime view of the MARVEL architecture for the GRN4-UJ3 scenario.

Table 3: MARVEL architectural components for GRN4-UJ3: MARVEL Data Corpus

MARVEL subsystem	Component information		
	Name	Owner	Functionality
Sensing and perception subsystem	GRNEdge	GRN	AV data collection and transmission
Security, privacy and data protection subsystem	Anonymisation	FBK	Video anonymisation offline version of VideoAnony component
Optimised E2F2C	Karvdash	FORTH	Service deployment

processing and deployment subsystem			
E2F2C infrastructure	Cloud layer	PSNC	Cloud layer for GRN4-UJ3
	Fog layer	GRN	Fog layer for GRN4-UJ3
Decision-making and user interactions subsystem	MARVEL Data Corpus-as-a-Service	STS	MARVEL Data Corpus

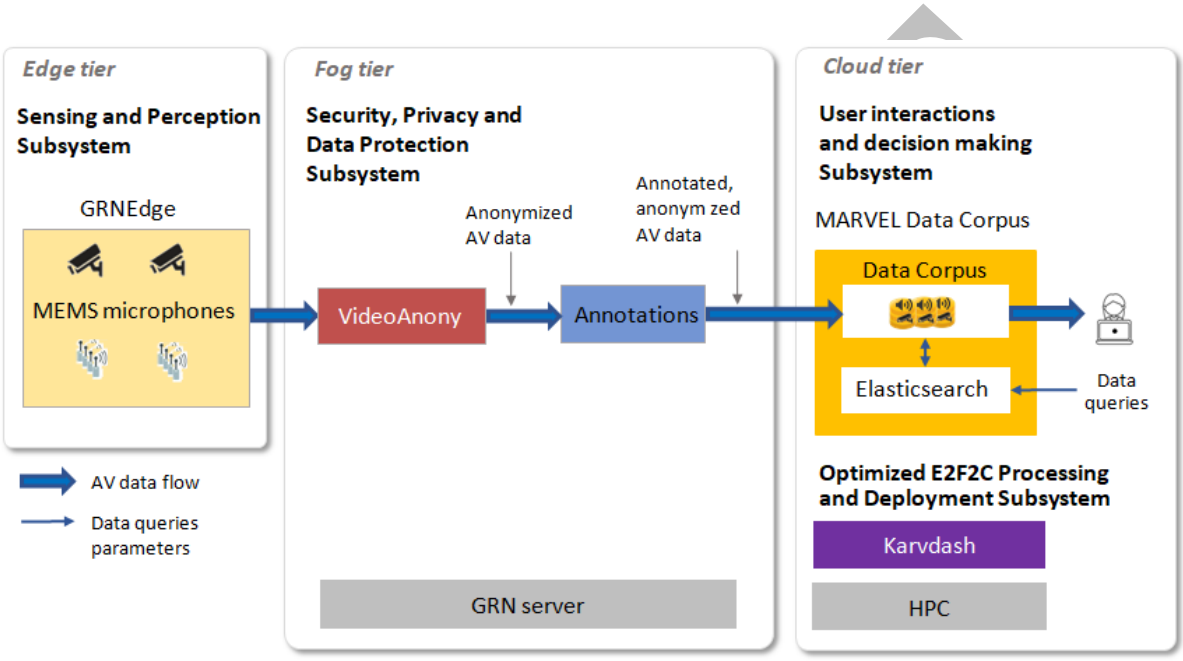

Figure 23. Scenario 3: MARVEL Data Corpus – Deployment and runtime view

4.1.4 Data management platform architecture

The Data Management Platform (DMP) figure which presents the overview of the data management and distribution subsystem, also showing the main component interactions and data flows, is illustrated in Figure 21 of MARVEL D1.3. For the sake of saving space, we omit including it here; the interested reader can refer to D1.3, p. 62[23]. For the purposes of the MVP, specific parts of the subsystem were selected and were refined and re-specified. Figure 24 illustrates the selected components. As shown in the Figure, the core DMP components for the MVP are the DFB and DatAna. Selected functionalities of those components serve as technological enablers towards the MVP. Specifically, Apache NiFi to allow the processing of data flows between the edge and the fog layers, Kafka for stream processing, and Elasticsearch. The data considered for the MVP will originate from GRN and will be distributed using CATFlow. The aforementioned component refinements are reflected in the selected data flows which are depicted as red arrows in Figure 24. The eventual data flows will arrive at the DMT components (WP4) for ultimate visualisations.

[23] "D1.3 Architecture definition for MARVEL framework," Project MARVEL, 2021. https://zenodo.org/record/5463897#.Yc19Sy8Rr0p

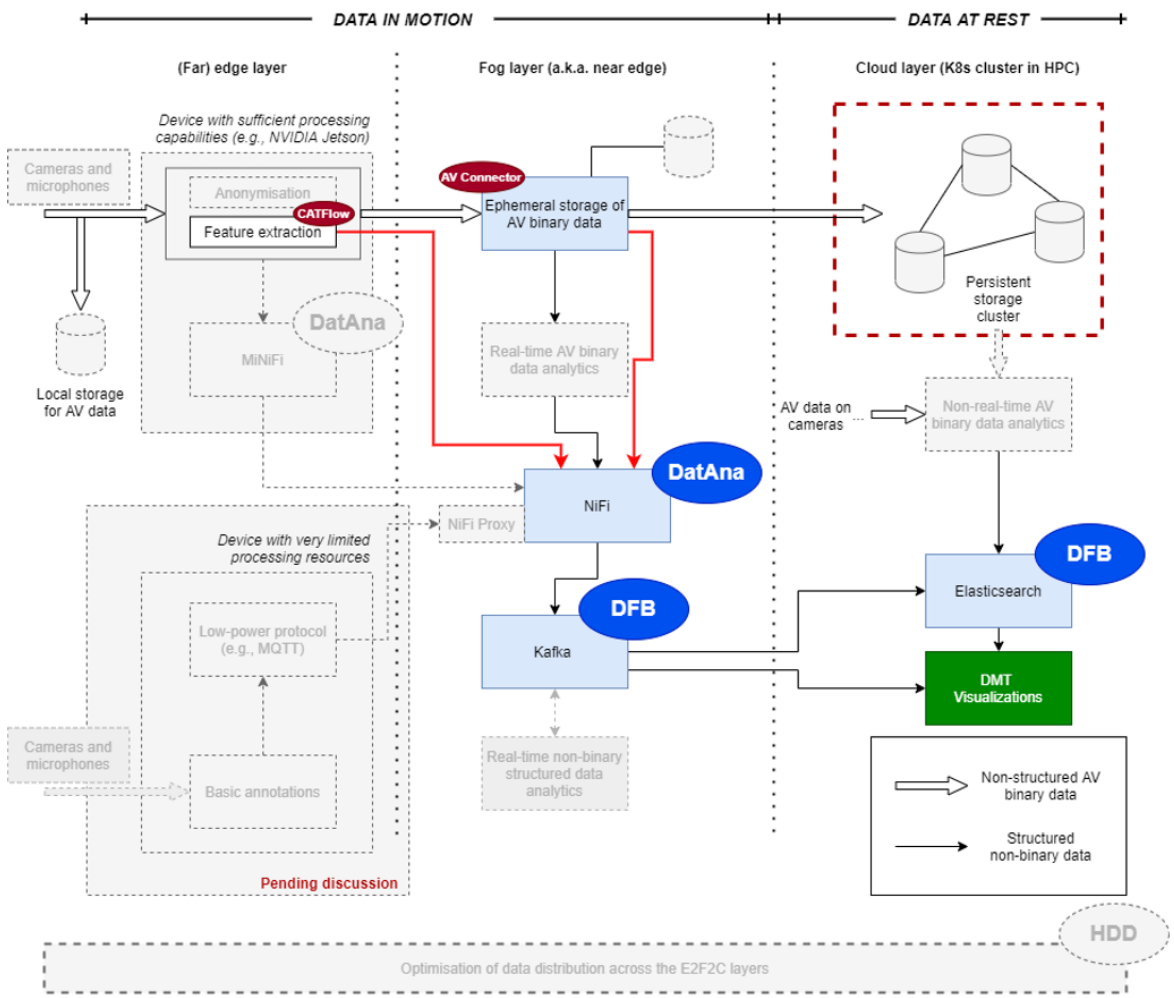

Figure 24. Selected DMP components for MVP refinement

The selected components from the GRN-DMP-DMT chain are extracted and illustrated in Figure 25. The exact data flows that are demonstrated in the scope of the MVP are shown in black arrows. The blue part represents the GRN components, while the orange parts represent the DMP and DMT components.

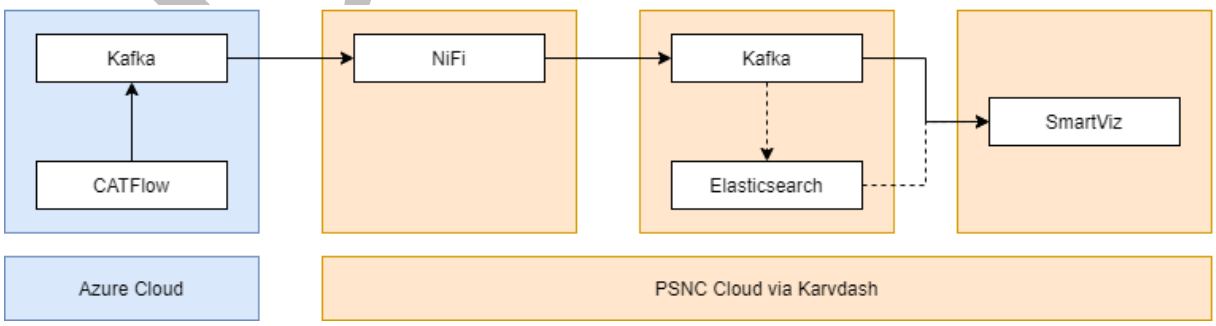

Figure 25. Selected DMP components' interactions for the MVP

In the following table, the implemented actions regarding the integration of the components in the GRN-DMP-DMT continuum are listed. On the left part of the table, pair of components

are reported (following the logical flow of Figure 25). On the right part of the table, the implemented actions are reported.

Table 4: Implemented actions for DMP components

Integration	Implemented actions
CATFlow – DatAna	• Azure connection • Azure Kafka topics for DatAna consumer • Data contents
DatAna – DFB	• DFB connection • DFB Kafka topics for DatAna producer • Elastic Indices • DatAna transformations • Data Schema for DFB input
DFB – SmartViz	• DFB Kafka topics for SmartViz consumer • Necessary queries to Elasticsearch

With regards to the integration of GRN's CATFlow and DatAna, technical details on Azure connections, Kafka topics for data consumers and data contents were addressed. With regards to the integration of DatAna and DFB, DFB connections, Kafka topics from data producers, Elastic indices, DatAna transformations and Data Schemas were addressed. With regards to the DFB - SmartViz integration, Kafka topics where SmartViz should subscribe for visualisations and Elasticsearch indices to query for historical data visualisations were addressed.

4.2 Infrastructure

PSNC cloud infrastructure

PSNC provides computing resources for various MARVEL subsystems operating in the cloud layer. The provided infrastructure consists of access to the HPC supercomputer and access to virtualised private cloud. The powerful supercomputer Eagle offers 1.4 PFLOPS of processing power. It consists of more than a thousand nodes with a total of more than 30 thousand cores and 120 TB of operational memory (RAM). Additionally, to take full advantage of the AI-based algorithms for AV analytics, the computing platform is equipped with 78 GPUs NVIDIA V100 with 32GB of memory each.

However, the primary role in MVP is played by virtualised private cloud resources. These are available via LabITaaS, which is the processing and data collection node, offering cloud-based services. LabITaaS is managed by OpenStack web interface and deployed as Infrastructure-as-a-Service. All functionality of OpenStack governed cloud is available via REST API allowing for building flexible flows that adjust used resources exactly to current needs. This approach enables scripted deployment and utilisation of auto-scaling and reliability mechanisms.

The first component used in the MVP cloud layer is Karvdash that belongs to the Optimised E2F2C processing MARVEL subsystem. Karvdash is installed on a VM equipped with 8 CPUs, 16 GB RAM, 512 GB of block storage, runs Ubuntu 20.04.1 LTS, and Kubernetes 1.19.8.

The next two components in the MVP cloud layer represent the Data management and Distribution Subsystems. First of them is DatAna, a simple NiFi service located in the PSNC

cloud and managed by Karvdash. DatAna acts as an entry point and interfaces with the second component of this subsystem that is DFB. Communication between these components is accomplished via Kafka topics.

The last components in the MVP cloud layer refer to the User interactions and decision-making subsystem. The goal of SmartViz is to provide advanced UI visualisations of the DMT. Additionally, PSNC provides VM and sufficient storage resources for the MVP version of MARVEL Data Corpus as-a-Service.

4.3 Components' deployment and integration

After having specified the views of the architecture and the size of the required infrastructure, we proceeded with the technical activities of deploying and integrating the selected components. The outcome of all activities is a functional release of the MVP, accessible to authorised users of the MARVEL platform[24].

The first activity deals with updating and shipping the individual components. As mentioned in Section 2, a GitLab is put in place for MARVEL partners to use. We decided that all software components are shipped as dockerised containers, thus providing a unified way of handling and deploying components, as well as resolving issues of dependencies, libraries and environment configuration for each component.

For all these components, we used Karvdash as the dashboard to execute and monitor component deployments.

For the data management and decision-making components (DatAna, DFB, and SmartViz), we proceeded with test integrations, as these components are connected and used in various ways for different scenarios. We established a pipeline in which DatAna retrieves some sample data from a file system, streams these data to DFB's Kafka on a test topic, DFB indexes the data in Elasticsearch, and finally SmartViz subscribes to the test topic, queries the Elasticsearch and presents real-time and historical data respectively with generated sample visualisations. At this stage, the proof-of-concept that we address is that this pipeline can be reused for different scenarios. The added value of this pipeline for scale and performance will be more evident in future releases where a number of components will stream large volumes of data to the data management platform for visualisation.

For Scenario 1 and Scenario 2, we defined the non-AV data formats that represent the information produced and handled for these specific scenarios. We have chosen JSON for all non-AV data exchanged between components. For Scenario 1, CATFlow's output contained vehicle detection and trajectory information (Appendix B), while for Scenario 2, the outputs of the AI components (AVCC, SED) were used to define the structure to be exchanged (Appendix C).

Overall, the three supported scenarios are implemented by the integrated components of the MVP, corresponding to the user journeys described in this document. The scenarios, the participating components, and the data flow description are explained in the remainder of this section.

***Scenario 1*:** The purpose of this scenario is to provide the end-user with information about detected vehicles and their trajectories. In a real-world operation, CATFlow would receive real-time AV data from cameras and operate on these data to detect vehicles and trajectories. For the MVP, CATFlow receives a set of files with recorded video and sound from the

[24] https://www.marvel-platform.eu

cameras on the street level. The results are retrieved by DatAna that, in its turn, streams the collected data to DFB's Kafka. The data are indexed in the Elasticsearch instance so as to be available at any time. Finally, SmartViz queries the Elasticsearch API, retrieves the data, and presents them to the end-user, as presented in detail in Section 5.1.

***Scenario 2*:** The purpose of this scenario is to present the end-user with detailed advanced information, including sound events and pedestrian distribution on the area of the monitored junction. The data flow for this scenario is similar to Scenario 1. The AV files with recorded video and sound from the street-level cameras are made available through the GRNEdge module to two AI components: SED and AVCC. The inference results of these components are, respectively, timestamps on the recorded video that mark an interesting sound event, and a heatmap of pedestrian distribution on the area of the junction. Similar to Scenario 1, these inference results are streamed through DatAna to DFB, and the indexed information is accessed and visualised by SmartViz, as detailed in Section 5.1.

***Scenario 3*:** The purpose of this scenario is to lay the path for the infrastructure that supports training of AI models, by forwarding AV data to the MARVEL Data Corpus. The Data Corpus component was successfully deployed. The recorded AV files with the accompanying metadata (timestamps, sources, file type, duration, etc.) were transmitted to the Corpus from the GRNEdge module, after passing through video anonymisation. The stored data can be monitored and queried by the Corpus admin dashboard.

The above scenarios were tested thoroughly in parts and in end-to-end tests. The tests were in general successful, after solving many technical and configuration issues that arose. In many cases, the inference results of the AI algorithms were not accurate, as these algorithms were trained with generic data or with training transfer techniques. However, the purpose of the MVP release is to showcase the potentials of the framework, leaving accuracy, efficiency and optimisation issues to next releases when larger quantities of relevant datasets will be available.

Additionally, the testing phase revealed several issues that need to be addressed in detail for the upcoming integrated releases of the MARVEL framework, as for example, the performance issues that will appear when the size of data becomes significant, the real-time version of the tested inputs, the establishment of links and references between AV and metadata (annotations, inference results), and the need to improve accuracy and efficiency of AI algorithms through training with appropriate datasets.

5 Demonstration

In this section, we present the MVP demonstrator that focuses on services provided from the end-user point of view. We describe the role of the Decision-Making Toolkit as a means for visualising the collected and processed traffic data and relevant detected events. For each user journey defined, we present the screens with which the end-user interacts.

5.1 The Decision-Making Toolkit

The Decision-Making Toolkit will be the user interface of the MARVEL project. For the scope of the MVP, we have decided to focus on the use case GRN4: Junction Traffic Trajectory collection (see section 3.2.2 for more details of the use case that has been described in detail in D1.2: MARVEL's Experimental Protocol).

Principles of Design Thinking were leveraged in order to design the UX of the MVP as well as the extended MARVEL solution. For the purposes of the MVP, we identified the user (traffic engineer) as well as their prioritised need which led to this use case in the first place: *"As a Transport engineer, I need a way to track and analyse traffic data in various junctions, so that I devise a better traffic plan for the area".*

We then identified a few ideas to tackle this need and we prioritised them according to impact and feasibility. The ones with the highest impact and feasibility are prioritised to be implemented and released as our MVP. The detailed user journey is described in paragraph 5.2 of this document. The ideas prioritised are:

1. The user will be able to view the trajectories of various types of vehicles on the junctions watched on a camera image (batch data).
2. The user will be able to view the number of different types of vehicles appearing in the junction on various interactive graphs (batch data).
3. The user will be able to view anomalies on the junctions watched on a map (real-time info with geolocation coordinates).

In later versions after the MVP, we are planning to position this information on the map as well in order to better satisfy this need and enrich the features covering this use case.

As a result of the process described above, the DMT will provide to the users of MARVEL visualisations of detected events mostly associated with traffic and vehicles. The toolkit will allow the visualisation for the facilitation of medium to long-term decisions with the assistance of the predefined widgets. A few preconfigured visualisation schemas were selected to address the ideas described earlier based on best practices and on ZELUS' team experience for user-friendliness and intuitiveness, in order to allow the use of the toolkit by users with no need for extensive training.

Users will firstly choose from the available use cases (Figure 26). In the context of the MVP, the GRN4: Junction Traffic Trajectory was the one selected for development.

Figure 26. SmartViz dashboard

The dashboard for this use case (Figure 26) allows users to choose one of the available scenarios per pilot use case. For the MVP, use case GRN4: Junction Traffic Trajectory Collection links to two implement scenarios, detailed in the following subsections.

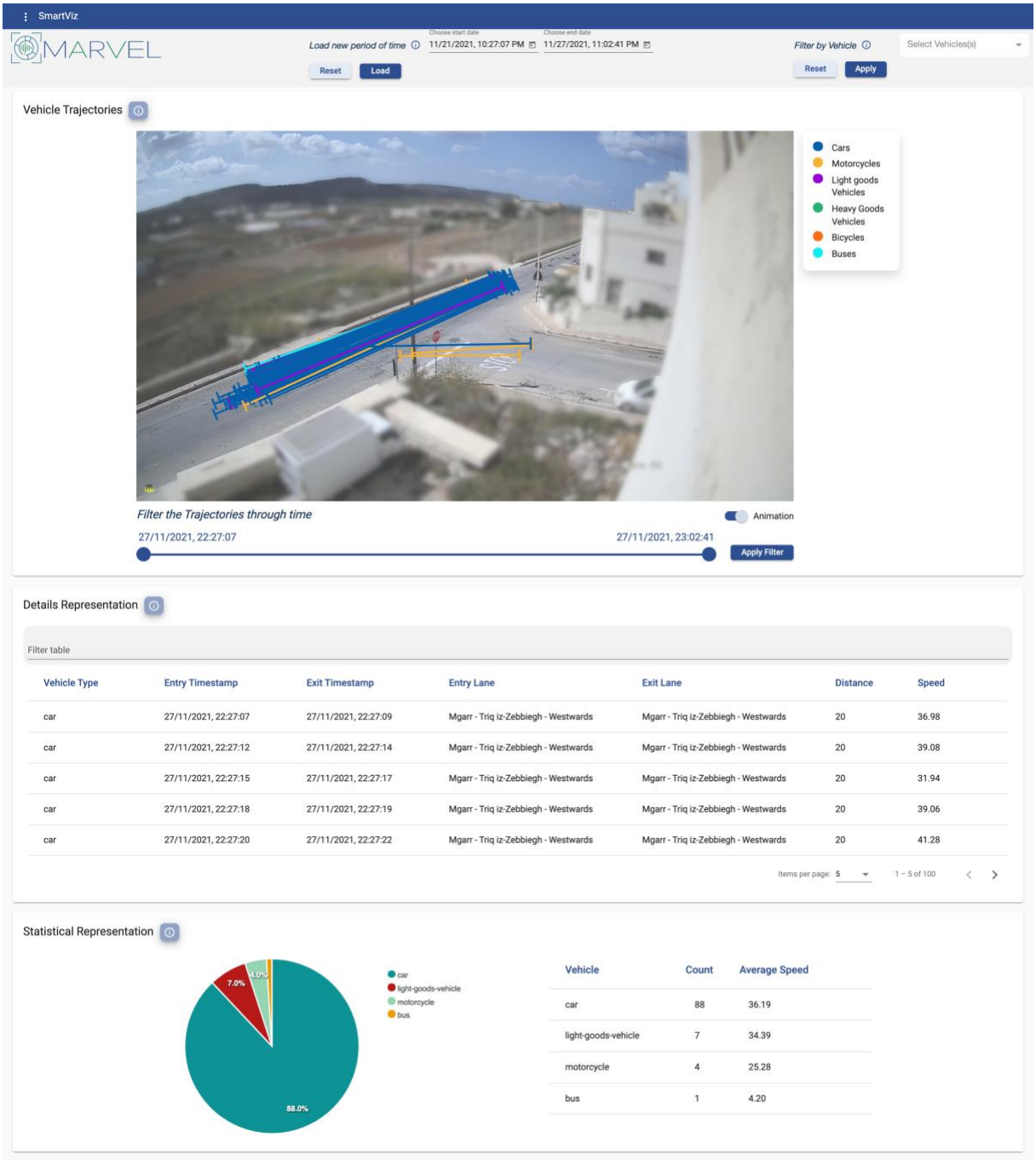

Figure 27. The trajectories widget

Detailed information regarding the trajectories of the detected vehicles will be presented in the Trajectories widget. In this widget, the paths of the passing vehicles are drawn in the image of the camera feed that is recording them. The paths are grouped and color-coded depicting different kinds of vehicles. The user is able to change the time period and select the type of vehicle to further investigate.

Detailed information about the detected incoming events will be presented in the Details widget. This widget also supports a standalone text filtering capability in order to search through the available information.

Although events coming to the DMT are only the ones produced by the GRN pilot and the AI system for the MVP, SmartViz is capable of supporting all the other sources that will be deployed in the full version of the MARVEL framework.

5.2 User journeys for GRN Use Case 4

5.2.1 User Journey 1: Vehicle and Bicycle Trajectories, and Vehicle counting

This user journey tackles the need for traffic engineers to view and analyse the trajectories of cyclists and other vehicles. This will allow the traffic engineers to make informed decisions on issues such as the most preferred location of a cycle path or if there needs to be more enforcement on issues such as heavyweight vehicles staying on the left side of the road. This data could then be visualised as a collection of traces drawn on an image of the road being investigated. The UI widget that the end-user interacts with is shown in Figure 27.

The edge component for this use case will be the Mgarr Camera. This camera can stream both audio and video, however, only video is required by the CATFlow system, which is used for the MVP implementation. The video data is streamed using the RTSP protocol to the GRN fog layer. Here the CATFlow algorithm is used to process the video. The output from the CATFlow algorithm is structured non-binary data, in JSON format. Included in this data are points that trace each vehicle's trajectory.

For the needs of the MVP, the JSON data produced by CATFlow are stored in files in Karvdash shared folders, from where DatAna takes them, provides the necessary formatting and passes the messages to the DFB Kafka in the cloud. The streamed data can be accessed and used for visualisation. For the purposes of visualisation, the data will be used to collect the trends of trajectories. Real-time analytics are not required in this user journey as decisions will be made on historical data collected.

Additionally, this user journey presents a count (a distribution) of the different types of vehicles passing through a road junction or segment. This will allow the user to be able to track how frequently a road is being used and by what type of vehicle. This information could then be used to make data-driven decisions on the road infrastructure. For example, a road frequently used by heavy vehicles would need to be re-surfaced more often and/or would require a different mix of road surfacing materials. Alternatively, the engineer might consider rerouting the heavy vehicles. On the other hand, a road frequently used by cyclists would make a candidate for the installation of a cycling lane or the installation of cycling infrastructural technology. For the MVP, only simple data processing algorithms are implemented such as collecting and separating the traffic counts based on the different types of vehicles or the visualisation of colour-coded trajectories which a human being would be able to visually process and reach conclusions.

Finally, the information of detected vehicles is also presented to the user in a temporal form, as shown in Figure 28. In the table shown, the horizontal axis is time, divided into slots of one hour. Vertically, the grouping of vehicles by type is shown. The detected vehicles, color-coded and labelled with their vehicle type, are represented as dots at the exact moment of detection.

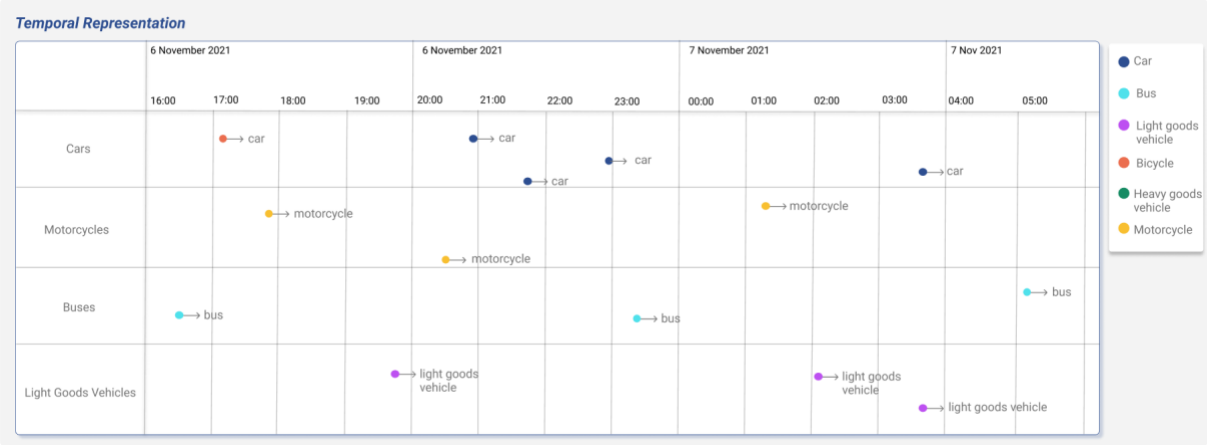

Figure 28. Temporal representation of detected vehicles

5.2.2 User Journey 2: Sound Events and Crowd counting

Similar to User Journey 1, the primary end-users for this user journey are traffic engineers that need to view and analyse advanced detected events, namely sound events and count of pedestrians on the junction area. While UJ1 presents information useful for monitoring a junction, this UJ draws the attention of the end-user to specific events and areas of the camera frame where a decision may need to be taken. In future releases, any relevant algorithm of the complete set of MARVEL's AI algorithms could be incorporated in this setting. In Figure 29, the UI screen for UJ2 is shown.

The edge component for this use case will be the Mgarr Camera. This camera can stream both audio and video, that are processed by the SED and AVCC components. SED outputs a JSON structure with a detected sound event type, and the timestamp for this occurrence. For example, one of the capabilities of SED is to identify a vehicle by its sound. AVCC outputs a JSON with points on each frame of a video with a likelihood value for a pedestrian to be present at that point. This information is summarised in a matrix that corresponds to the input frame and later visualised as a heatmap over the original frames. Examples of both outputs of the two components are presented in Appendix B and Appendix C.

As in UJ1, the inference results in JSON format are stored in files in Karvdash shared folders. DatAna accesses those folders and after providing the necessary formatting, it passes them as JSON messages to the DFB Kafka in the cloud, then indexed in the Elasticsearch module and can be accessed and used for visualisation. An example of a visualisation of the heatmap for detected pedestrians is shown in Figure 29.

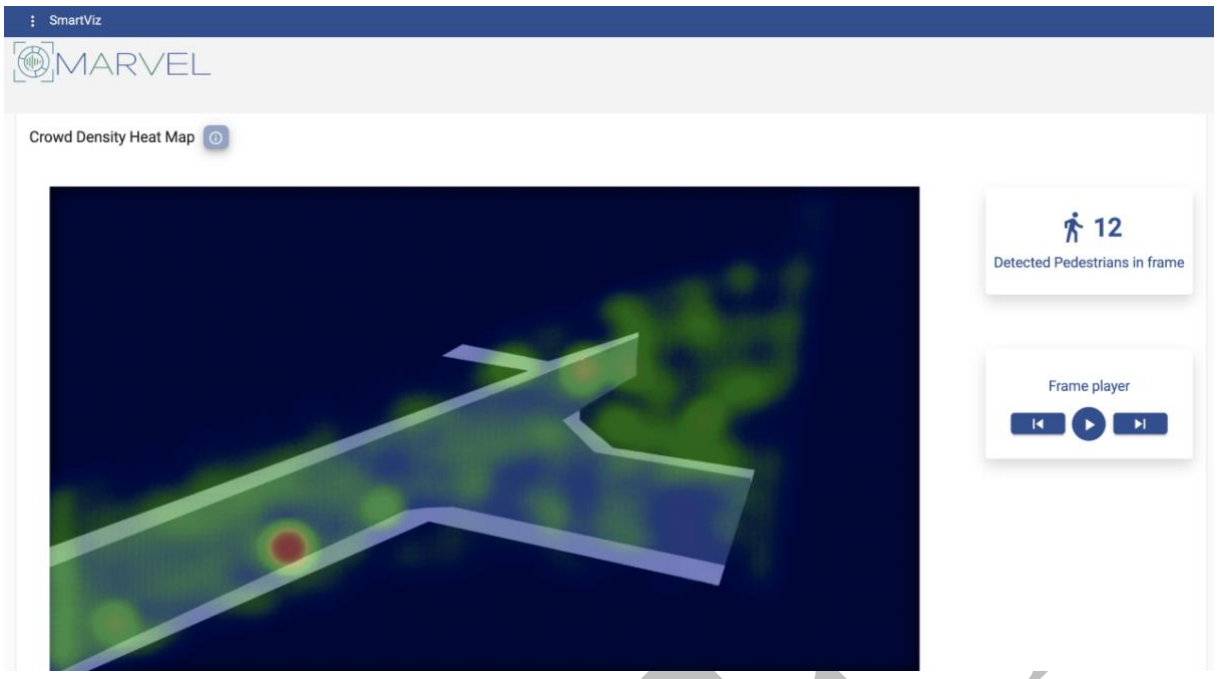

Figure 29. Pedestrian heatmap screen for UJ2

5.2.3 User Journey 3: Populating the MARVEL Data Corpus

Unlike the previous UJs, this user journey is not directly targeted to traffic engineers or any other professional end-users that need to monitor events on a street level and make a decision. For this UJ, the end goal is to collect, and store in the Data Corpus all annotated, anonymised AV data that will be available to the AI components for model training and potentially to any other client that may use these data. At this stage, as well as during the phase of growth and maintenance of the Data Corpus, the targeted end-users for this UJ are administrators that need to monitor both the contents of the Corpus (statistics for files, details of individual files), as well as monitoring the health and performance of the underlying infrastructure. In Figure 30 and Figure 31, we can see two administration screens that show a master-detail view of an uploaded file and the admin dashboard view, respectively.

The value of these UI elements will be more evident as the Data Corpus size will increase in the course of the project, as content and infrastructure monitoring activities will be crucial for the performance of the Data Corpus.

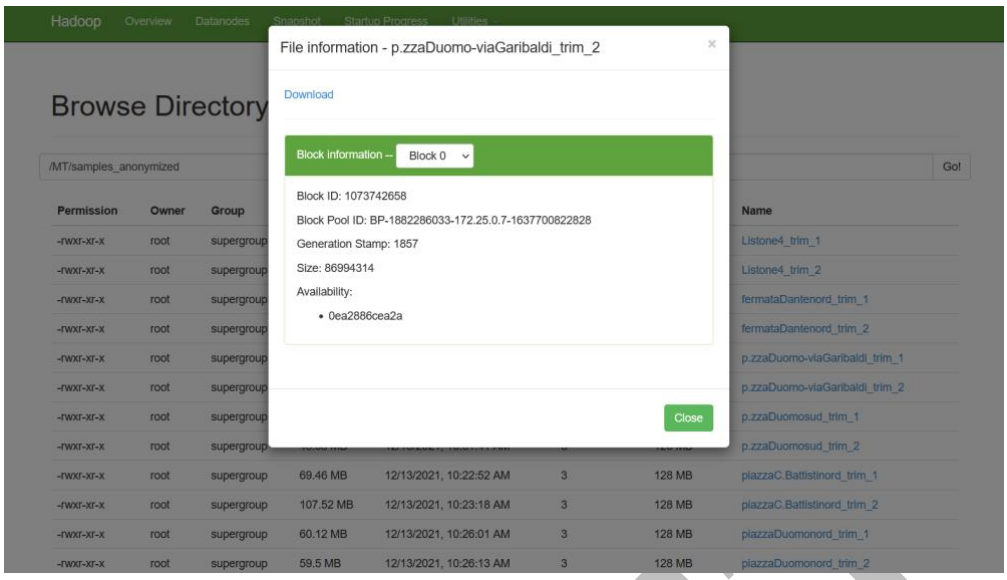

Figure 30. Master-detail view of the AV files stored in the Data Corpus

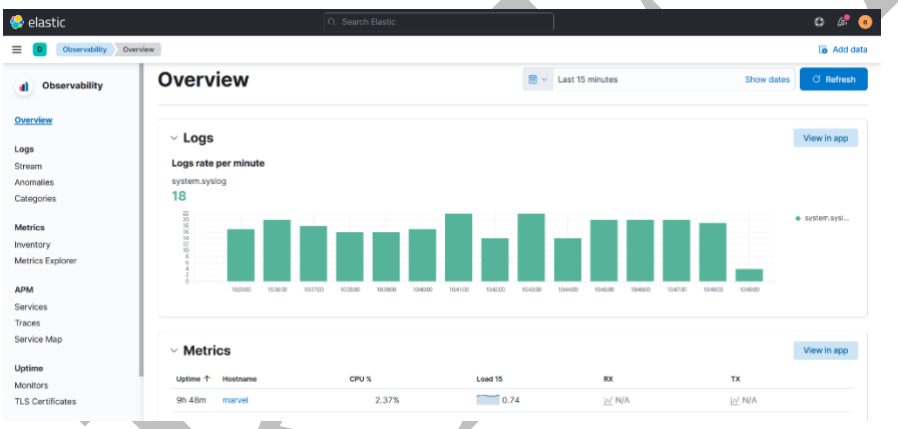

Figure 31. Administration dashboard for the Data Corpus

6 Contribution to MARVEL goals

The overarching goal of MARVEL is to deliver an Edge-to-Fog-to-Cloud (E2F2C) framework that operates in real-world environments, processing large volumes of captured AV data, enabled by multi-modal perception and intelligence. More specifically, with regards to the project objectives analysed in the DoA, the delivery of the MVP is related to 'Overall project Objective 3 that states:

> *Objective 3: Break technological silos, converge very diverse and novel engineering paradigms and establish a distributed and secure Edge-to-Fog-to-Cloud (E2F2C) ubiquitous computing framework in the big data value chain Challenge Big Data phenomenon, often called a 'data tsunami' raises multifaceted technological challenges. AI platforms, like IoT applications, rely their success not only on efficient data processing/management but also in a coordinated action where diverse technological silos are converged. In contrast to other domains where these actions have been established (e.g., connected cars), multimodal perception in a smart city environment lacks significantly behind.*

The MVP presented in this document contributes to this objective by setting the stage for the E2F2C infrastructure, incorporating a set of AI components that showcase their perception and intelligence capabilities, as well as demonstrating successful end-to-end scenarios for data management and presentation. This contribution can be regarded as "setting a steppingstone" towards the upcoming, complete versions of MARVEL that will expand the scale of the MVP in terms of infrastructure, provided services and volume of data to be processed. Finally, the proof-of-concept demonstration of the MVP validates the value delivered to the end-user in a controlled environment and lays the ground for next releases to operate and be tested in real-life experiments.

The MVP also addresses the enablers that are linked to the above objective, namely a) E1-HPC infrastructure and resource management, b) E2-Distributed and optimised DL models deployment, c) E3-Secure and distributed computing framework, and d) E4-Complex decision-making and insights. In this document, we have presented a minimum subset of MARVEL technologies that address the above-mentioned topics of end-to-end infrastructure, distributed AI, distributed computing, and decision-making capabilities. These issues will be further developed and explored in the next releases of the framework.

Finally, the MVP provides a functional platform that can be used for the evaluation of technical KPIs that are linked to the above Objective (e.g., KPI-O3-E1-1 to KPI-O3-E4-1), concerning novel algorithms, algorithmic accuracy and performance, and facilitation of complex decision-making. A thorough technical evaluation of the MVP will take place during the upcoming benchmarking activities (Task 5.4), while end-user feedback and evaluation will occur during the scheduled events and Info-days. The results of these evaluations will be detailed in the respective deliverables D6.2, D6.3, D6.4, D5.2, and D5.5.

7 Conclusions

In this document, we presented the MVP for the MARVEL framework. We embarked from the previous work on framework architecture and use case definition and proceeded with the specification of the scope and goals of the MVP. We presented in detail the process and rationale for selecting the appropriate use case to be demonstrated and the technical decisions regarding infrastructure, user journeys, participating components, and integration of technologies into a unified, end-to-end platform. We discussed results and observations from the development and operation of the MVP and linked this release to the overall MARVEL objectives and subsequent releases.

This document can be used as a basis and reference for planned activities, deliverables and milestones, most notably: a) the planned benchmarking sessions within Task 5.4, to be part of deliverable 'D5.2 - Technical evaluation and progress against benchmarks – initial version', b) the public events, showcases and info days where the MVP will be demonstrated to the public, c) the pilot execution and evaluation, deliverables 'D6.1 – Demonstrators execution – initial version', 'D6.2 – Evaluation report', and d) the upcoming complete versions of the integrated framework, deliverables 'D5.4 - MARVEL Integrated framework – initial version', 'D5.6 - MARVEL Integrated framework – final version'.

Especially for the above-mentioned initial version of the MARVEL integrated framework (M18), it constitutes a natural continuation of the integration efforts that have started with the MVP. To that end, we plan to set a detailed roadmap towards M18, for designing, implementing and delivering a complete version of MARVEL of greater scale and operated in a real-world setting. In terms of integration processes, as explained in this document, we plan to enrich our CI/CD process with more issue tracking, documentation and automation tools that will be necessary for managing the expected data volume, number of components and underlying infrastructure.

Bibliography

[1] Eric Ries. “The Lean Startup: How Today's Entrepreneurs Use Continuous Innovation to Create Radically Successful Businesses”. New York: Crown Business, 2011.

[2] Jez Humble, David Farley. "Continuous Delivery: Reliable Software Releases through Build, Test, and Deployment Automation”. Addison-Wesley Signature Series 1st Edition

[3] Qiuqiang Kong, Yin Cao, Turab Iqbal, Yuxuan Wang, Wenwu Wang, and Mark D. Plumbley. "Panns: Large-scale pretrained audio neural networks for audio pattern recognition." IEEE/ACM Transactions on Audio, Speech, and Language Processing 28 (2020): 2880-2894.

[4] Pablo Zinemanas, Pablo Cancela, and Martin Rocamora, "MAVD: A Dataset for Sound Event Detection in Urban Environments", Proceedings of the Detection and Classification of Acoustic Scenes and Events 2019 Workshop (DCASE2019), pages 263–267, New York University, NY, USA, Oct. 2019

[5] Jakob Abeßer, Saichand Gourishetti, András Kátai, Tobias Clauß, Prachi Sharma, and Liebetrau, IDMT-Traffic: An Open Benchmark Dataset for Acoustic Traffic Monitoring Research, EUSIPCO, 2021.

[6] Su, Xiu, et al. "Vision transformer architecture search." arXiv preprint arXiv:2106.13700 (2021).

Appendix A

Data Corpus data model

```
{
"id": "<Entity's unique identifier>",
"category": "<String Reference that points to the type of data, e.g. Audio, Video, Img>",
"dataProvider": "<String Reference that points to the data pilot, e.g. UNS, GRN, MT>",
"description": "<A text-based description of the data>",
"use_case": "<String Reference that points to the corresponding usecase, e.g. GRN1>",
"dataset": "<String Reference that points to the corresponding dataset described in D2.1>",
"keywords": "<A set of keywords describing the topic of the data>",
"video": {
        "video_metadata": {
                "video_resolution": "<the video resolution>",
                "video_fps": "<video frames per seconds>",
                "annotation_type": {
                        "annotation_software": "",
                        "annotation_ontology": ""
                },
                "device_id": "<points to file that gives device info>",
                "Location": {
                        "latitude": "<the latitude of the recording>",
                        "longitude": "<the longitude of the recording>"
                },
                "duration": "<video duration in seconds>",
                "total_snippets": "<the total number of snippets>",

                "snippet": [
                        {
                                "id": "<snippet unique identifier>",
                                "publication_date": "<Date of data creation>",
                                "duration": "<snippet duration in seconds>",
                                "starts": "<snippet starting point in seconds>",
                                "ends": "<snippet ending point in seconds>",
                                "timestamp": "<the timestamp of data ingestion>",
                                "Datanode_HDFS": "<name of datanode where snippet is stored>",
                                "Url_HDFS": "<the url/path where the snippet is stored>",
                                "is_annotated": "<flag that marks if data is annotated>",
                                "annotator_id": "List of annotators",
                                "annotation_files": "a list of annotation files",
                                "annotation_summary":  "<type of events that refer to the
corresponding snippet, parse the annotation file -> to place here>",
                                "additional_events": "extra annotation events",
                                "Augmentation": {
                                   "is_augmented": "<flag that marks if snippet is augmented>",
                                    "augmentation_method":  "<Enumerator that points to the
augmentation technique>"
                                },
                                "Anonymization": {
                                  "is_anonymized": "<flag that marks if data is anonymized>",
                                   "anonymization_method":  "<Enumerator that points to the
anonymized method>"
                                }
                        }
                ]
        }
},
"audio": {
        "audio_metadata": {
                "audio_bitrate": "<the audio bitrate>",
                "audio_sampling": "<sampling frequency in Hz>",
                "no_of_channels": "<the number of channels>",
                "annotation_type": {
                        "annotation_software": "",
                        "annotation_ontology": ""
                },
                "Location": {
                        "latitude": "<the latitude of the recording>",
                        "longitude": "<the longitude of the recording>"
                },
                "duration": "<audio duration in seconds>",
```

```
                "total_snippets": "<the total number of snippets>",

                "snippet": [
                        {
                                "id": "<snippet unique identifier>",
                                "publication_date": "<Date of data creation>",
                                "duration": "<snippet duration in seconds>",
                                "starts": "<snippet starting point in seconds>",
                                "ends": "<snippet ending point in seconds>",
                                "timestamp": "<the timestamp of data ingestion>",
                                "Datanode_HDFS": "<the name of the datanode where snippet is
stored>",
                                "Url_HDFS": "<the url/path where the snippet is stored>",
                                "is_annotated": "<flag that marks if data is annotated>",
                                "annotator_id": "List of annotators",
                                "annotation_file": "<the url/path where corresponding annotation
file is stored>",
                                "annotation_summary": "<type of events that refer to the
corresponding snippet>",
                                "additional_events": "extra annotation events",
                                "Augmentation": {
                                        "is_augmented": "<flag that marks if snippet is
augmented>",
                                        "augmentation_method": "<Enumarator that points to the
augmentation technique>"
                                },
                                "Anonymization": {
                                        "is_anonymized": "<flag that marks if data is
anonymized>",
                                        "anonymization_method": "<Enumarator that points to the
anonymized method>"
                                }
                        }
                ]
        }
},

"audio-video": {
        "audio-video_metadata": {
                "video_resolution": "<the video resolution>",
                "video_fps": "<video frames per seconds>",
                "audio_bitrate": "<the audio bitrate>",
                "audio_sampling": "<sampling frequency in Hz>",
                "annotation_type": {
                        "annotation_video_software": "the sofware type used for video
annotation",
                        "annotation_audio_software": "the sofware type used for audio
annotation",
                        "annotation_video_ontology": "the ontology used for video annotation",
                        "annotation_audio_ontology": "the ontology used for audio annotation"
                },
                "device_id": "<points to file the gives device info>",
                "Location": {
                        "latitude": "<the latitude of the recording>",
                        "longitude": "<the longitude of the recording>"
                },
                "duration": "<audio-video duration in seconds>",
                "total_snippets": "<the total number of snippets>",

                "snippet": [
                        {
                                "id": "<snippet unique identifier>",
                                "publication_date": "<Date of data creation>",
                                "duration": "<snippet duration in seconds>",
                                "starts": "<snippet starting point in seconds>",
                                "ends": "<snippet ending point in seconds>",
                                "timestamp": "<the timestamp of data ingestion>",
                                "Datanode_HDFS": "<the name of the datanode where snippet is
stored>",
                                "Url_HDFS": "<the url/path where the snippet is stored>",
                                "is_annotated": "<flag that marks if data is annotated>",
                                "annotator_id": "List of annotators",
                                "annotation_files": "a list of annotation files",
                                "annotation_summary": "<type of events that refer to the
corresponding snippet>",
                                "additional_events": "extra annotation events",
```

```
                                "Augmentation": {
                                        "is_augmented":   "<flag   that   marks   if   snippet   is
augmented>",
                                        "augmentation_method":  "<Enumerator  that  points  to  the
augmentation technique>"
                                },
                                "Anonymization": {
                                        "is_anonymized":   "<flag   that   marks   if   data   is
anonymized>",
                                        "anonymization_method": "<Enumerator that points to the
anonymized method>"
                                }
                        }
                ]
        }
}
```

Appendix B

CATFlow output


```
{
    "msg_ver": "1.0.0",
    "app_ver": "1.0.0",
    "type": "vehicle-cam",
    "id": "6e389259-657e-4002-96c1-e4079db1d867",
    "sent_ts": "2021-11-17T13:02:21.144054+01:00",
    "sys_name": "Greenroads",
    "sys_id": "greenroads",
    "node_id": "ABC",
    "seq_id": "20211117130216-greenroads-streams1-stream1",
    "seq_no": 1,
    "location": {
        "name": "Mgarr - iz-Zebbiegh",
        "uuid": "f630a888-6f40-4784-b377-e1c75f2b849b",
        "lat": "35.91877777777778",
        "lon": "14.376527777777778"
    },
    "camera": {
        "type": "CAM",
        "group_id": "Mgarr iz-Zebbiegh",
        "loc_id": "Mgarr - iz-Zebbiegh",
        "cam_id": "greenroads-streams1-stream1",
        "cam_name": "greenroads-streams1-stream1",
        "cam_uuid": "af19f507-5306-4e19-9f30-f271b4971c9f",
        "lat": "35.91877777777778",
        "lon": "14.376527777777778"
    },
    "vehicle": {
        "type": "car",
        "name": "Car",
        "entry": {
            "ts": "2021-11-17T13:02:17.863805+01:00",
            "carriageway_name": "Mgarr - Triq iz-Zebbiegh",
            "carriageway_uuid": "ba2baff8-4e9d-4cee-8b05-6ae0662b42bf",
            "lane_name": "Mgarr - Triq iz-Zebbiegh - Eastwards",
            "lane_uuid": "a55a5b77-ecca-47fc-850a-ed0bcab28574",
            "lane_type": "general",
            "lane_flow_uuid": "ecab2998-f218-4413-b3bf-e403c244eeb4",
            "uuid": "a66a40d5-0f97-4290-a9ad-fe88f0714eab",
            "midpoint": {
                "x": 0.5848,
                "y": 0.4259
            }
        },
        "exit": {
            "ts": "2021-11-17T13:02:19.393100+01:00",
            "carriageway_name": "Mgarr - Triq iz-Zebbiegh",
            "carriageway_uuid": "ba2baff8-4e9d-4cee-8b05-6ae0662b42bf",
            "lane_name": "Mgarr - Triq iz-Zebbiegh - Eastwards",
            "lane_uuid": "a55a5b77-ecca-47fc-850a-ed0bcab28574",
            "lane_type": "general",
            "uuid": "11c6e77a-0d84-4ff2-8cd6-343b7e430ceb",
            "midpoint": {
                "x": 0.218,
```

```
                    "y": 0.7685
                },
                "lane_flow_uuid": "ecab2998-f218-4413-b3bf-e403c244eeb4"
            },
            "distance_meters": 15,
            "time_seconds": 1.529295,
            "speed_kmh": 35.310388120016086,
            "trajectory_points": [
                [
                    0.5471354166666667,
                    0.42592592592592593
                ],
                [
                    0.18671875,
                    0.6842592592592592
                ]
            ]
        }
}
```

Appendix C

SED output


```
{
    "results": [
        {
            "onset": 2.0,
            "offset": 4.0,
            "label": "car"
        },
        {
            "onset": 3.0,
            "offset": 5.0,
            "label": "car"
        },
        {
            "onset": 4.0,
            "offset": 6.0,
            "label": "car"
        },

            […]

}
```


AVCC data


```
{
  "Data" : {
    "Frame" : 0,
    "predicted_count" : "23.0",
    "image_paths" :
"../data/sample_crowd/Video_for_Crowd_counting/frames/0.jpg",
    "audio_paths" :
"../data/sample_crowd/Video_for_Crowd_counting/audio/0.wav",
    "density_paths" : null,
    "inference_time" : "3.05",
    "description" : "Crowd counting detection",
   "id" : "303d3f27-1f77-4d74-b573-3080d733d7ac",
   "type" : "MediaEvent",
    "eventType" : "CrowdCountingEvent",
   "mediaSource" : {
       "filename" :
"../data/sample_crowd/Video_for_Crowd_counting/greenroads-streams1-
stream12021-11-16_06-27-45+0000.mkv",
```

```
        "creationTime" : "2021-12-10T11:39:12+0000",
        "name" : "AVCC - Audio Visual Crowd Counting"
    },
   "dateCreated" : "2021-12-16 07:30:05 ",
    "Heatmap" : [
      […]
     ]
}
```