PANTHEON

Community-Based Smart City Digital Twin Platform for Optimised DRM operations and Enhanced Community Disaster Resilience

D4.3

Enhanced Intelligence & Self-adaptive Simulations

The project has received funding from the European Union's Horizon Europe programme under Grant Agreement N°101074008.

DOCUMENT INFO

Deliverable Number	D4.3
Work Package Number and Title	4 Design and Development of a Smart City Digital Twin for Community DRM
Lead Beneficiary	SIMAVI
Due date of deliverable	28/02/2025
Deliverable type[1]	OTHER
Dissemination level[2]	PU
Author(s)	Iacob Crucianu (SIMAVI); Otilia Bularca (SIMAVI)
Internal reviewer(s)	Mike Karamousadakis (THL), Anna Tsabanakis (THL)
Version - Status	**1.0 Final**

TASK ABSTRACT

The deliverable incorporates the results of Task T4.3-Enhanced Intelligence & Self-adaptive Simulations related to Community Disaster Management Models. According to the GA (PANTHEON - Consortia, 2023), this task investigates the development and integration of Artificial Intelligence algorithms to provide the simulation models with the capability of self-adaptation. Self-adaptation allows the model to adjust relevant parameters based on the real data received and the evaluation of its performance against goals set in the different scenarios. Furthermore, this task explores Machine Learning methods and approaches to provide the simulation models with the capability of monitoring their behaviours and collecting the needed data and information to have sufficient situational awareness. Also, the task investigates Machine Learning approaches to analyse this data and provide the simulation.

The main focus of this deliverable is on Artificial Intelligence algorithms that provide simulation models with the capability of self-adaptation based on the statistical models that are described in D4.1.

[1] **Please indicate the type of the deliverable using one of the following codes**:
R = Document, report
DEM = Demonstrator, pilot, prototype, plan designs
DEC = Websites, patents filing, press & media actions, videos
DATA = data sets, microdata
DMP = Data Management Plan
ETHICS: Deliverables related to ethics issues.
OTHER: Software, technical diagram, algorithms, models, etc.

[2] **Please indicate the dissemination level using one of the following codes**:
PU = Public
SEN = Sensitive

REVIEW HISTORY

Version	**Date**	**Modifications**	**Editor(s)**
0.1	11/11/2024	First draft. TOC	Iacob Crucianu (SIMAVI); Otilia Bularca (SIMAVI)
0.2	15/11/2024	General approach	Iacob Crucianu (SIMAVI); Otilia Bularca (SIMAVI)
0.3	04/02/2025	Wildfire use case	Iacob Crucianu (SIMAVI); Otilia Bularca (SIMAVI)
0.4	05/02/2025	Earthquake use case	Iacob Crucianu (SIMAVI); Otilia Bularca (SIMAVI)
0.5	07/02/2025	Heatwave use case	Iacob Crucianu (SIMAVI); Otilia Bularca (SIMAVI)
0.6	08/02/2025	Man-made disaster scenario	Iacob Crucianu (SIMAVI); Otilia Bularca (SIMAVI)
0.7	14/03/2025	Remarks on scenarios	Mike Karamousadakis
0.8	19/03/2025	Details for an earthquake scenario	Iacob Crucianu (SIMAVI); Otilia Bularca (SIMAVI)
0.9	21/03/2025	Details for heatwave scenario	Iacob Crucianu (SIMAVI); Otilia Bularca (SIMAVI)
0.10	27/03/2025	Reformatting, Rewriting formulas	Iacob Crucianu (SIMAVI); Otilia Bularca (SIMAVI)
1.0	02/04/2025	Final version	Anna Tsabanakis (THL)

DISCLAIMER

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Description
AI	Artificial Intelligence
API	Application Programming Interface
CBDRM	Community-based Disaster Risk Management
CSV	Comma Separated Values
DAG	Directed Acyclic Graphs
FR	Functional Requirement
GIS	Geographic Information System
HTML	HyperText Modeling Language
IT	Information Technology
IUC	Intended Use and Classification
JSF	Java Server Faces
JMS	Java Messaging System
JSON	JavaScrip Object Notation
LAT	Latitude
LONG	Longitude
LPDM	Lagrangian Particle Dispersion Model
LSTM	Long Short-Term Memory
ML	Machine Learning
NFR	Non-functional Requirement(s)
RL	Reinforcement Learning
SCDT	Smart City Digital Twin
UAV	Unmanned Aerial Vehicle

EXECUTIVE SUMMARY

Background	Goals
To enable the definition and usage of Digital Twins, the Conceptual models are analysed and presented in this deliverable. The conceptual models will focus on the management of multi-hazards and critical system interactions, and they will be part of SCDT developments. Systems simulating multi-hazards are very complex, with numerous interacting components. The conceptual model simplifies these complexities by focusing on the key elements and their relationships, allowing us to grasp how the system functions without getting overwhelmed by every detail.	This Deliverable aims to describe the Conceptual Models used so that the development of individual components can take place in WP4, WP5 and WP6 along with the integration of the platform in WP7. Specifically, the work presented here builds upon the work done in D2.4, D3.1, D3.2, D3.3, D3.4 and D3.6, by further specifying (i) the necessary system requirements to fulfil the user requirements described in D3.2 and D3.6 and (ii) the technical components that participate in the architecture along with the sequence diagrams of their interconnection after analysis on (a) the existing technological landscape (D2.4 & D3.1) (b) the conceptual model of the SCDT (D3.3), (c) the data delivery schemes in D3.4 and (d) the definition of data in D4.1.

Approach and course of action
Utilise work achieved in previous work packages and previous tasks: • WP2 PANTHEON Approach for Building Disaster-Resilient Communities • WP3 PANTHEON Requirements, Participatory Design Process and Pilot Use-Cases Specifications First, the theoretical background is presented for the models considering the specific scenarios and use cases. Next, the SCDT as the main system approach is described, and the conceptual models are presented.

Findings and results
Four categories of models are described and proposed to be used in PANTHEON: • Wildfire models • Earthquake models • Heatwave models • Man-Made Disaster models

Impact	Planned dissemination and exploitation
Conceptual models for multi-hazard management are analysed and the most appropriate for the use cases in Pantheon are proposed. These models set the groundwork for the technical development of the PANTHEON platform in WP4, WP5, WP6 and WP7.	Public.

1. INTRODUCTION

1.1 OVERVIEW

The current deliverable, D4.3 – Enhanced Intelligence & Self-Adaptive Simulations, presents the results of the research and development conducted in the corresponding task (T4.3). The work focused on exploring Artificial Intelligence (AI) and Machine Learning (ML) algorithms to enable self-adaptation in disaster simulation models. By integrating these techniques, the simulation models can dynamically adjust their parameters based on real-time data and continuously evaluate their performance against predefined goals.

The work presented in this document builds upon the findings and methodologies anticipated in *T4.1 Big Data generation and harvesting* where the core data models and integration mechanisms for simulation models are defined and on *T4.2 Analysis and representation of collected data & Conceptual Models* that develops the technical framework and software architecture for executing these simulations. To demonstrate the proposed techniques and methods, the outcomes of *D3.6 Use Case Scenarios* where considered, mainly the list of use cases and real-world disaster scenarios, that provide essential requirements for developing self-adaptive simulations.

Moreover, T4.3 investigates the development and integration of AI-driven self-adaptation mechanisms in simulation models for disaster risk management (DRM) and the objectives include:

- Developing AI-driven self-adaptive mechanisms that enable simulation models to autonomously adjust their parameters based on real-world data;
- exploring ML techniques for real-time data analysis and decision-making, allowing models to determine when and how to adapt;
- optimizing disaster response strategies by integrating reinforcement learning, anomaly detection, and predictive analytics into simulations;

The proposed approach to achieve these objectives integrates graph theory, queue theory, and statistical modelling for a mathematical foundation for self-adaptive simulations:

- **Graph Theory** is used for network-based disaster modelling, particularly in evacuation planning and emergency response routing and depicts road networks, infrastructure dependencies, and resource distribution as nodes and edges, enabling path optimization and rerouting strategies.
- **Queue Theory** is applied to model traffic congestion, hospital triage systems, and emergency response bottlenecks while simulating queue dynamics in evacuation scenarios, medical resource allocation, and transportation delays.
- **Statistical Modelling** uses Poisson, Weibull, Pareto, and Exponential distributions to model disaster event frequencies, resource demands, and emergency response times.

Following this approach, the current deliverable extends the capabilities of static simulation models into dynamic, adaptive systems that evolve alongside changing disaster scenarios, improving the effectiveness of disaster management strategies.

1.2 DELIVERABLE STRUCTURE

The document titled “D4.3 – Enhanced Intelligence & Self-adaptive Simulations” is part of Work Package 4, activities focus on the Design and Development of a Smart City Digital Twin for Community Disaster Risk Management (DRM) and explores the integration of AI and ML algorithms for self-adaptive simulation models to improve disaster resilience.

The document is structured into two main sections: one introducing the general approach and another one presenting the AI-enhanced self –adaptive simulation models.

The g**eneral approach** section explains the methodology for designing AI-enhanced self-adaptive simulations and describes the processing flow for real-time monitoring, adaptation, and decision-making.

Next, the **AI-Enhanced Self-Adaptive Simulations** section provides an overview of the AI techniques used to enhance simulations. The deployed methods and algorithms describe statistical models, graph theory, queue theory, and reinforcement learning approaches.

To demonstrate the proposed models, PANTHEON use case scenarios are used and the proposed simulation covers detailed simulations covering different disaster scenarios: (i) wildfire models, (ii) earthquake models, (iii) heatwave models and (iv) man-made disaster models.

Overall, the document presents a conceptual model based on: (i) mathematical models for AI-driven adaptation, (ii) graph-based representations for transportation and disaster management, (iii) queue theory simulations for optimizing response times and traffic management and (iv) REST API descriptions for integrating the simulation system with external platforms.

Along the way, several academic papers, software tools, and relevant frameworks are referred to.

2. GENERAL APPROACH

2.1 OVERVIEW

The complexity and frequency of disasters—both natural and man-made—have highlighted the need for robust, adaptive simulation models capable of supporting real-time decision-making. Simulation models for scenarios such as wildfires, earthquakes, heatwaves, and man-made disasters are essential tools for predicting disaster progression, assessing risk, and aiding response efforts. However, these models must operate in highly dynamic environments where conditions can shift unpredictably due to factors like changing weather, structural vulnerabilities, and human behaviour. Traditional static simulation models struggle to keep pace with these rapid changes, often requiring manual adjustments and recalibration to maintain accuracy.

The current approach proposes exploring the role of AI in enhancing the adaptability of the conceptual simulation models, presented in D4.2 (PANTHEON Consortia, 2024) for various types of disasters. By incorporating self-adaptive mechanisms, AI-driven simulation models can automatically adjust parameters, refine predictions, and improve their responses based on real-time data. Self-adaptive simulations use advanced AI techniques—including Bayesian optimization, reinforcement learning, and anomaly detection—to modify key parameters dynamically, thereby allowing simulations to reflect the latest environmental, structural, or contextual changes. This adaptability transforms simulations from static tools into dynamic systems capable of evolving alongside the situations they model.

Each type of disaster presents unique modelling challenges, requiring specialized approaches to adaptation. For instance, wildfire simulations need to account for changing wind conditions, fuel loads, and moisture levels, while earthquake models must respond to tectonic shifts and evolving aftershock patterns. Similarly, heatwave simulations need to incorporate fluctuations in temperature, humidity, and urban heat island effects, whereas models for man-made disasters, must adjust to varying parameters specific to the disaster considered.

This section includes a comprehensive overview of AI methods and algorithms that enable self-adaptation in these simulation models. It details how specific AI techniques are applied to each disaster type, describing the underlying models, adaptive workflows, and example scenarios where self-adaptation enhances predictive accuracy and operational utility. Additionally, it addresses key implementation considerations, including data integration or real-time processing constraints.

2.1.1 KEY CHARACTERISTICS OF AN EFFECTIVE SELF-ADAPTATION MECHANISM

Self-adaptive mechanisms describe the capacity of a system to change its behaviour under conditions that can occur at runtime. Several characteristics are relevant to demonstrate its efficacy:

- Real-time responsiveness refers to the capability of the mechanism to make adaptations as soon as relevant data indicates a need for change, while also ensuring that the simulation remains accurate and relevant.
- Autonomy and minimal human intervention reflect the ideal situation when self-adaptation requires minimal human oversight, relying on AI to detect, decide, and execute changes autonomously.
- Contextual awareness represents the situational awareness that the system needs to adapt based on both internal performance metrics and external conditions (e.g., environmental factors or operational changes).

- Incremental and controlled adjustments suggest that adaptations should be incremental and controlled to prevent over-adjustment or instability, especially in complex systems where multiple parameters interact.
- Traceability and rollback capability states that every adaptation should be logged, allowing for post-analysis, traceability, and rollback in case an adaptation leads to suboptimal results.

2.1.2 PERFORMANCE FRAMEWORK

During the process of elaborating the AI-driven self-adaptive simulations, a performance framework was considered. First, the critical KPIs for the simulations are identified. These refer to efficiency, accuracy, processing speed, resource consumption, and other domain-specific metrics. Next, parameters that represent the model's state, such as input-output relationships, parameter values, and intermediate states are defined and finally, the thresholds and acceptable ranges for each metric to recognize when the model's performance deviates from expected behaviour were established.

2.2 PROCESSING FLOW

The AI-driven self-adaptive simulation framework follows a structured process flow applicable to all four disaster scenarios: wildfire, earthquake, heatwave, and man-made disasters. This approach ensures real-time monitoring, adaptation, and decision-making, optimizing evacuation, resource allocation, and emergency response.

A stepwise approach is proposed for the processing flow as shown in the figure below.

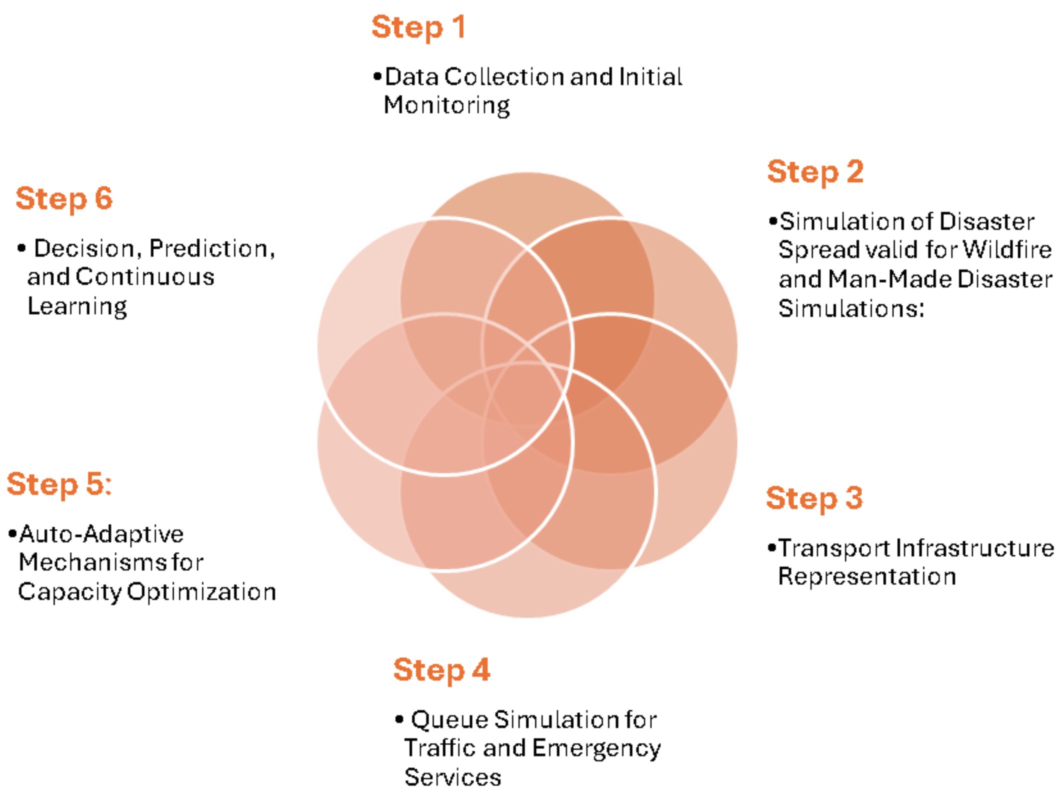

Step 1
•Data Collection and Initial Monitoring
Step 2
•Simulation of Disaster Spread valid for Wildfire and Man-Made Disaster Simulations:
Step 3
•Transport Infrastructure Representation
Step 4
• Queue Simulation for Traffic and Emergency Services
Step 5:
•Auto-Adaptive Mechanisms for Capacity Optimization
Step 6
• Decision, Prediction, and Continuous Learning

Figure 1 Processing flow approach

Step 1: Data Collection and Initial Monitoring

The system continuously monitors and collects real-time data from sensors, weather stations, infrastructure reports, and emergency call logs. Data is stored in a specific storage system, and for our system is exported in csv or JSON format, used as input for simulation.

Additional manual input is entered to characterize specific environmental conditions (e.g., fire intensity, toxic gas properties, infrastructure resilience). Some input is interactive during the simulations.

Step 2: Simulation of Disaster Spread valid for Wildfire and Man-Made Disaster Simulations:

Fire or toxic gas dispersion models simulate how hazards spread based on wind speed, temperature, and terrain. Impact assessments determine how infrastructure, roads, and buildings are affected. The results influence traffic flow and emergency response planning.

Step 3: Transport Infrastructure Representation

The road network is modelled as a **graph**, where nodes represent intersections, and edges represent roads, with attributes like capacity, length, and congestion levels. Shortest paths and independent paths are calculated to optimize evacuation and emergency response. Multiple scenarios are simulated, where one or more roads or intersections are removed to test resilience and re-routing strategies.

Step 4: Queue Simulation for Traffic and Emergency Services

M/M/c queue models simulate congestion at intersections (traffic flow under evacuation conditions), hospitals and emergency centres (patient intake rates during a disaster event).

Step 5: Auto-Adaptive Mechanisms for Capacity Optimization

Real-Time Reconfiguration of Nodes and Edges: AI dynamically modifies node capacities (e.g., increasing lane usage at key intersections). Edges (roads) are adjusted to reflect lane reassignments and emergency-only routes.

Queue Length Reduction Strategies: Rule-Based Adjustments modify road priorities and hospital intake distributions. Reinforcement Learning (RL) Algorithms learn from past scenarios, improving future response efficiency.

Step 6: Decision, Prediction, and Continuous Learning

Scenario-Based Decision Making: AI recalculates the best evacuation paths, hospital capacities, and emergency response strategies based on changing conditions.

Feedback and Learning: The system stores optimized configurations for road networks and emergency response plans to improve preparedness.

2.2.1 WORKFLOW OF THE SELF-ADAPTATION MECHANISM

The workflow design presented below starts from the KPI and thresholds definition, continues with data collection and monitoring, adds a behavioural and performance analysis, launches the decision process, executes the commands, refines, evaluates and continues in feedback loops.

1. **KPIs and thresholds** definition for model prediction accuracy establish what “success” looks like for the simulation and set acceptable error bounds for predictions.

2. **The data collection and monitoring** mechanism begins with monitoring real-time inputs and the simulation's internal performance metrics. The data collected reflects the current state and any relevant environmental or contextual factors influencing the simulation.
3. **The behavioural and performance analysis** module evaluates the monitored data. It assesses whether the current model settings meet the simulation goals or if performance is deteriorating due to external changes. The analysis might involve comparing historical trends, running predictive models, or using situational awareness to understand external impacts.
4. **The decision process** launches when analysis identifies deviations from expected outcomes or recognizes an opportunity for improvement, the decision-making module evaluates options for adaptation. The module uses optimization or machine learning techniques to decide the type and extent of adaptation, balancing different priorities to avoid over-adjusting.
5. **The execution and update** step is active after a decision is made, and the adaptation execution module implements the changes within the simulation model. This could involve tuning specific parameters, switching between model configurations, or updating underlying decision algorithms to reflect new priorities.
6. **Evaluation and Feedback Loop**: After adaptation, the mechanism continues to monitor the simulation to evaluate whether the adjustments yield the desired results. This forms a feedback loop, where the system continually learns from its adaptations, improving future decisions and adaptability.

2.2.2 PLACE IN THE OVERALL PANTHEON ARCHITECTURE

The algorithms used for the self-adaptive simulation process are part of the overall architecture (Figure 2 The functional view of the PANTHEON architecture) and cover Data Analysis for ML algorithms, Machine Learning Models and simulation Models.

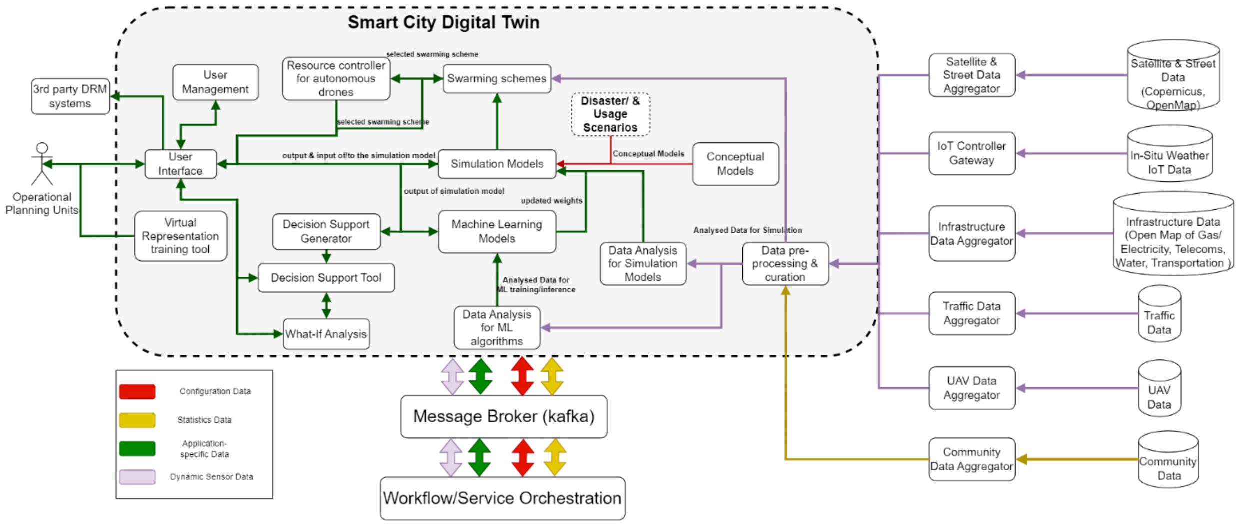
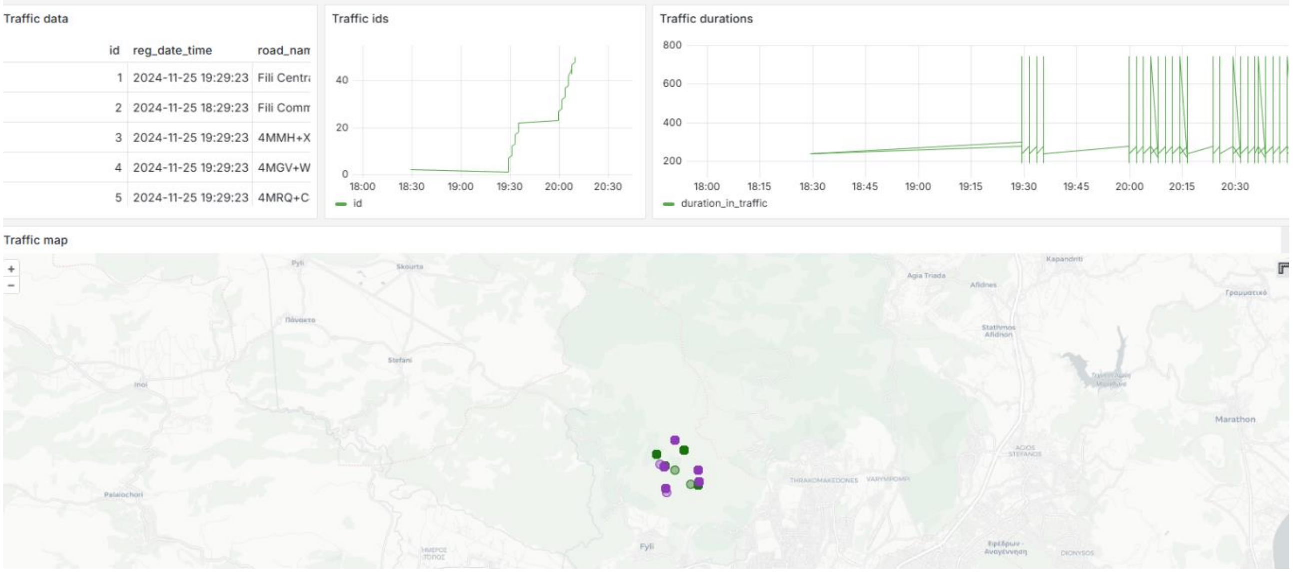

Smart City Digital Twin
3rd party DRM systems
User Management
Resource controller for autonomous drones
selected swarming scheme
Swarming schemes
selected swarming scheme
Disaster/ & Usage Scenarios
Operational Planning Units
User Interface
output & input of/to the simulation model
Simulation Models
Conceptual Models
Conceptual Models
output of simulation model
updated weights
Virtual Representation training tool
Decision Support Generator
Machine Learning Models
Analysed Data for Simulation
Data Analysis for Simulation Models
Data pre-processing & curation
Decision Support Tool
Analysed Data for ML training/inference
What-If Analysis
Data Analysis for ML algorithms
Configuration Data
Statistics Data
Application-specific Data
Dynamic Sensor Data
Message Broker (kafka)
Workflow/Service Orchestration
Satellite & Street Data Aggregator
Satellite & Street Data (Copernicus, OpenMap)
IoT Controller Gateway
In-Situ Weather IoT Data
Infrastructure Data Aggregator
Infrastructure Data (Open Map of Gas/ Electricity, Telecoms, Water, Transportation)
Traffic Data Aggregator
Traffic Data
UAV Data Aggregator
UAV Data
Community Data Aggregator
Community Data

Figure 2 The functional view of the PANTHEON architecture

It is important to mention that the input data used comes from different sources presented in D4.2 ((PANTHEON Consortia, 2024)), is heterogeneous and should be normalized.

In the first stage, data is monitored and collected in the format it is on the source. Next, it is normalized and placed in csv of JSON format, available for the scope of PANTHEON. The data collected and used is from satellites (Copernicus) and local authorities.

An example of a data monitoring tool used to collect geographical data is presented below.



Traffic data

id	reg_date_time	road_nam
1	2024-11-25 19:29:23	Fili Centr
2	2024-11-25 18:29:23	Fili Comm
3	2024-11-25 19:29:23	4MMH+X
4	2024-11-25 19:29:23	4MGV+W
5	2024-11-25 19:29:23	4MRQ+C

Figure 3 Data monitoring and visualisation

Data captured is then placed in the specific format required by running the algorithms, and this will be presented for each particular case in the next chapters.

3. AI ENHANCED SELF-ADAPTIVE SIMULATIONS

3.1 OVERVIEW

The self-adaptation mechanism in simulation models functions as a feedback-based system, using real-time monitoring, data analysis, and AI-based decision-making to adaptively tune model parameters. This adaptability allows simulation models, especially those powered by Digital Twins, to respond dynamically to changing conditions, ensuring they continue to meet performance goals and remain relevant as the environment evolves.

AI-enhanced self-adaptive simulations leverage Statistics, Graph Theory, Queue Theory, and AI-based models to dynamically adjust response strategies for disasters such as wildfires, earthquakes, heatwaves, and man-made crises. These methods allow for real-time decision-making, optimizing resource allocation, traffic management, and emergency response while continuously adapting to evolving conditions.

3.2 AI METHODS AND ALGORITHMS FOR SELF-ADAPTIVE SIMULATIONS

3.2.1 STATISTICAL MODELS

Statistical distributions [(Catherine Forbes, 2011; Davison, 2008)], play a critical role in queue simulation and disaster modelling, allowing the prediction of traffic congestion, resource demands, and emergency response times. The distributions selected for wildfire, earthquake, heatwave, and man-made disaster scenarios help simulate real-world uncertainties in traffic flows, emergency calls, and hospital triage processes. The most relevant statistical distributions include Poisson, Weibull, Pareto, and Exponential models, each offering unique advantages in modelling different aspects of disaster response.

3.2.1.1 *Theoretical Background of Key Distributions*

3.2.1.1.1. Poisson Distribution

The Poisson distribution models the probability of a given number of events occurring in a fixed time or space interval, assuming events occur independently.

Defined by the probability mass function:

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

Where λ is the average rate of occurrences.

Relevance to Disaster Simulations:

- Models emergency call rates for ambulance dispatch during heatwaves.
- Simulates arrival of vehicles at intersections in evacuation routes.
- Predicts the frequency of aftershocks following an earthquake.

3.2.1.1.2. Weibull Distribution

The Weibull distribution is widely used for reliability analysis and failure prediction, making it ideal for modelling infrastructure resilience in disaster scenarios.

Defined by the probability density function:

$$f(x;k,\lambda) = \frac{k}{\lambda}\left(\frac{x}{\lambda}\right)^{k-1} e^{-(x/\lambda)^k}$$

Where κ is the shape parameter and λ is the scale parameter.

Relevance to Disaster Simulations:

- Models failure rates of infrastructure (bridges, power lines) in earthquake and wildfire scenarios.
- Simulates fire spread behaviour by analysing burn rates of vegetation.
- Predicts evacuation time distributions, adjusting routes dynamically in heatwave and earthquake evacuations.

3.2.1.1.3. Pareto Distribution

The Pareto distribution is used to model extreme events, making it well-suited for resource distribution and rare but high-impact disaster events.

Defined by the probability density function:

$$f(x;x_m,\alpha) = \frac{\alpha x_m^{\alpha}}{x^{\alpha+1}}$$

Where x_m is the minimum value and α is the shape parameter.

Relevance to Disaster Simulations:

- Models traffic congestion in urban areas, where a small number of roads carry most of the traffic.
- Analyses resource shortages, as a small number of hospitals may receive the majority of emergency patients during a heatwave or earthquake.
- Predicts the probability of extreme fire outbreaks in wildfire scenarios.

3.2.1.1.4. Exponential Distribution

The Exponential distribution models the time between independent events, making it highly relevant for response times and inter-arrival processes.

Defined by the probability density function:

$$f(x;\lambda) = \lambda e^{-\lambda x}, \quad x \geq 0$$

Where λ is the rate parameter.

Relevance to Disaster Simulations:

- Models time intervals between emergency response dispatches (e.g., ambulance arrivals at hospitals during a heatwave).
- Predicts arrival rates of evacuees at triage centres.

3.2.2 GRAPH THEORY APPLIED TO TRAFFIC OR CRITICAL NETWORK REPRESENTATION

Graph theory [(Berge, 1982); (Diestel, 2017)], is a fundamental mathematical framework used in modelling disaster scenarios such as wildfires, earthquakes, heatwaves, and man-made disasters. A graph consists of nodes (vertices) and edges (connections between nodes) that together represent complex systems, including transportation networks, resource distribution, and emergency response logistics.

By applying shortest path algorithms, independent paths, queue modelling, and adaptive AI strategies, the system ensures effective traffic control, resource allocation, and emergency response in wildfire, earthquake, heatwave, and man-made disaster scenarios.

3.2.2.1 *Key Components of Graph Theory used:*

Nodes (Vertices):

Nodes represent key locations such as intersections, emergency stations, hospitals, fire outbreaks, and population centres. In traffic and evacuation modelling, intersections are modelled as nodes to analyse congestion and optimize routing. In healthcare response modelling, nodes represent triage stations and hospitals, helping allocate emergency medical resources effectively.

Edges (Connections Between Nodes):

Edges represent roads, pathways, or links between key locations. Each edge is assigned properties such as length, travel time, congestion level, or availability. In wildfire and earthquake scenarios, edges can be removed or blocked to simulate damaged infrastructure and reroute traffic accordingly.

Cycles (Loops in the Graph):

A cycle occurs when a sequence of edges forms a closed loop, allowing multiple routes between nodes. While cycles can provide alternative evacuation routes, they may also lead to inefficiencies if critical intersections become bottlenecks. In disaster simulations, cycles are removed or reduced to create simpler models that prioritize the most efficient paths.

Paths (Sequences of Connected Nodes and Edges):

A path is a sequence of nodes and edges that allows movement from one point to another. Shortest paths are computed to find the fastest evacuation routes or quickest access to hospitals. In earthquake and wildfire scenarios, path recalculations help adjust for blocked roads or changing fire conditions.

Independent Paths (Multiple Paths Without Shared Nodes):

Independent paths are essential in disaster management to prevent single points of failure. They ensure redundancy in the network by providing alternative evacuation routes or alternative transport paths for resources. In heatwave scenarios, independent paths help ambulances reach hospitals efficiently, preventing congestion at key intersections.

3.2.3 QUEUE THEORY APPLIED TO THE SIMULATIONS

Queueing theory (Lee, 1966) (Donald Gross, 2008) is a mathematical framework used to analyse waiting lines or queues. It studies the behaviour of systems where entities (e.g., data packets, tasks, or customers) arrive for service, wait in line if the service is busy, and eventually receive service. In telecommunications, the queueing theory provides tools to model and predict traffic behaviour in networks, where data packets compete for limited resources like bandwidth, processing power, or storage (Lee, 1966).

3.2.3.1 *Key Components of Queueing Theory used:*

1. Arrival Rate (**λ**): The rate at which packets arrive in the system.
2. Service Rate (**μ**): The rate at which packets are processed or transmitted.
3. Queue Discipline: The order in which packets are served (e.g., FIFO, priority-based).
4. Traffic Load (**ρ**): Defined as $\rho=\lambda/\mu$, this measures the utilization of the system.

Queueing theory uses various models (e.g., M/M/1, M/M/c) to analyse the system's performance under different conditions, such as single or multiple servers, finite or infinite queue lengths, and arrival/service distributions.

A model M/M/c contains the following elements:

- **Arrival Process (M):** Markovian (Poisson) arrival process with exponential interarrival times.
- **Service Process (M):** Markovian (exponentially distributed) service times.
- **Servers (c):** Multiple parallel servers (c).

A model M/M/1 uses the same arrival and process but only one server. It is suitable for analysing individual network components under light traffic, such as single routers, switches, or links. This is out of the scope of our paper which focuses on multiple redundant paths.

In our simulation, we will use the M/M/c model. These models are scenarios where packets are handled by multiple servers, such as in load-balanced routers or distributed data centres. It also helps evaluate the effect of increasing or decreasing the number of servers on performance.

3.2.4 REINFORCEMENT LEARNING (RL) FOR ADAPTIVE DECISION-MAKING

Auto-adaptive AI leverages Reinforcement Learning (RL) (Lapan, 2020)to optimize disaster response strategies dynamically. Unlike static models, RL-based AI systems continuously learn and adapt to real-time changes, making them ideal for complex and unpredictable scenarios such as wildfires, earthquakes, heatwaves, and man-made disasters. This framework enables intelligent decision-making in traffic management, resource allocation, and emergency response operations.

The particular type of RL used is based on Q-Learning algorithm (Lapan, 2020).

3.2.5 KEY CONCEPTS IN AUTO-ADAPTIVE AI USING RL

State Representation:

- Defines the current environment, including **traffic conditions, fire spread, seismic damage, hospital capacity, and resource availability**.
- Encapsulates real-time sensor data and predictive analytics to provide an accurate representation of the disaster landscape.

Action Space:

The set of possible actions the AI agent can take, such as:

- **Adjusting intersection capacities** to reduce traffic congestion.
- **Re-routing emergency vehicles** based on road conditions.

- **Allocating firefighting resources dynamically** to optimize suppression efforts.
- **Modifying hospital triage and admission rates** to balance patient loads.

Reward Function:

Determines the effectiveness of an action in achieving disaster response goals, such as:

- **Minimizing queue lengths** in evacuation scenarios.
- **Maximizing ambulance efficiency** in transporting patients.
- **Reducing fire spread and optimizing containment efforts**.
- **Ensuring hospitals maintain optimal capacity levels** without overwhelming staff and resources.

Policy Optimization:

- Uses algorithms such as **Deep Q-Networks (DQN)**, and **Proximal Policy Optimization (PPO)[** (Lapan, 2020)**]** to refine decision-making.
- The AI agent iteratively learns the best strategies to **improve response times, reduce casualties, and optimize resource deployment**.

Q-Learning algorithm (Lapan, 2020)

The key concepts used are:

- State (s): The current situation or configuration of the environment.
- Action (a): A possible move or decision the agent can take from a state.
- Reward (r): The immediate feedback received after taking an action.
- Q-value (Q(s, a)): The expected cumulative reward for taking action a in state s, and following the optimal policy thereafter.
- Learning Rate (α): Determines how much new information overrides old information.
- Discount Factor (γ): Determines the importance of future rewards.

The core of Q-learning is the Bellman Equation update:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_{a'} Q\left(s_{t+1}, a'\right) - Q(s_t, a_t) \right]$$

Where:

s_t: current state

a_t: action taken

r_{t+1}: reward

s_{t+1}: next state

$\max_{a'} Q\left(s_{t+1}, a'\right)$: maximum Q-value for the next state

3.2.6 APPLICATION OF THEORETICAL APPROACHES IN DISASTER SIMULATIONS

Wildfire Simulation:

- **Graph Construction:** Nodes represent key points (fire locations, safe zones, road intersections), while edges represent road connections.
- **Path Computation:** Shortest paths and independent paths are computed to **evacuate people safely**.
- **Queue Theory Integration:** Congestion at intersections is analysed using **queue models**, and reinforcement learning adapts routes dynamically.
- **Fire Spread Modelling:** Graph nodes and edges are dynamically updated to represent areas affected by fire, influencing traffic decisions.
- **RL dynamically adjusts** evacuation routes **based on fire spread projections.**
- **Weibull** distribution models the spread of fire over different terrain types.
- **Pareto** distribution predicts extreme fire intensities in regions with dense vegetation.

Earthquake Simulation:

- **Graph-Based Infrastructure Mapping:** Nodes represent **bridges, roads, buildings, and shelters**; edges represent transportation links.
- **Impact Analysis:** Earthquake damage is modelled by **removing damaged nodes and edges**, forcing recalculations of evacuation paths.
- **Auto-Adaptive Traffic Response:** Rule-based and RL-based **queue adaptations** adjust intersection capacities for efficient evacuations.
- **RL optimizes** traffic signal timing, reroutes emergency vehicles, and clears congestion **in affected areas, AI adapts road capacities based on** seismic damage data **and RL-based simulation predicts** bridge collapses, landslides, and transportation failures**.**
- **Weibull** distribution models infrastructure failures like building collapses.

Heatwave Simulation:

- **Traffic Flow for Ambulances:** Nodes represent **ambulance depots, intersections, triage centres, and hospitals**.
- **Multi-Level Graph Representation:**
- **Level 1 (Incident Location) → Level 2 (Triage Centers) → Level 3 (Hospitals)**
- **Queue-Based Simulations:** The model evaluates hospital and triage capacity, ensuring patients are efficiently allocated to medical facilities.
- **Adaptive Resource Allocation:** AI-based adjustments optimize patient distribution across hospitals to prevent **overcrowding**.
- **AI models simulate** ambulance dispatch efficiency by adjusting vehicle routing dynamically**.**
- **RL** fine-tunes hospital triage capacities to prevent bottlenecks in emergency rooms. Adjusts cooling centre accessibility based on real-time heat stress levels.
- **Poisson** distribution predicts ambulance call volumes during heatwave peaks.
- **Pareto** distribution models overloaded hospitals, where a small number handle most cases.
- **Exponential** distribution estimates the time between hospital admissions due to heat-related illnesses.

Man-Made Disaster Simulation:

- Gas spread using LPDM algorithm.
- **Critical Infrastructure Protection:** Graph theory models power grids, communication networks, and water supply systems.
- **Traffic and Evacuation Modelling:** Emergency response paths are dynamically adjusted to prevent bottlenecks.
- **RL minimizes congestion** in mass evacuation scenarios**,** ensuring safe and rapid exit**.** AI adjusts public transportation schedules and road access for optimal emergency response and dynamically adapts resource reallocation as the crisis unfolds.
- **Poisson** distribution models incident occurrence rates, such as industrial accidents.
- **Pareto** distribution models resource shortages, ensuring efficient emergency supply distribution.

3.3 STOCHASTIC WILDFIRE MODELS

We are using stochastic model for fire spread simulation which also allows users to interactively input wind speed, wind direction, and fire spread parameters. The system also models terrain and vegetation characteristics as static parameters while enabling real-time firefighting interventions through user input.

The fire spread model predicts the movement of fire based on:

- **Wind Speed and Direction**: Determines the rate and direction in which the fire advances.
- **Surface Type and Vegetation**: Areas with dense vegetation burn faster, whereas barren land slows fire spread.
- **Weather Conditions:** Changes in humidity, temperature, and precipitation affect fire intensity and behaviour.

The simulation uses a cellular automaton model, where each grid cell represents a section of land with specific properties, including burn potential and spread probability. The model is continuously updated based on new wind and environmental data, simulating how fire evolves.

3.3.1 KEY FACTORS IN WILDFIRE SIMULATION

An essential aspect of the simulation is the deployment and management of firefighting resources. The model evaluates different strategies such as:

- **Firebreaks**: Creating gaps in vegetation to halt the fire's progression.
- **Controlled Burns**: Using pre-emptive burning to eliminate fuel sources ahead of the fire front.
- **Aerial Fire Suppression**: Deploying water bombers and helicopters to target high-intensity areas.
- **Ground Crews:** Strategically placing firefighting teams based on terrain accessibility and fire intensity.

The model used is stochastic, so, to simulate the probabilistic nature of fire behaviour, different statistical distributions such as Poisson, Weibull, and Pareto (Catherine Forbes, 2011) are used. These distributions help model variations in fire spread speed and randomness in ignition probabilities. (See Figure 5 Wildfire simulationFigure 4)

3.3.2 USE CASES

The use cases specific to the wildfire scenario are presented in the next figure extracted from D3.6.

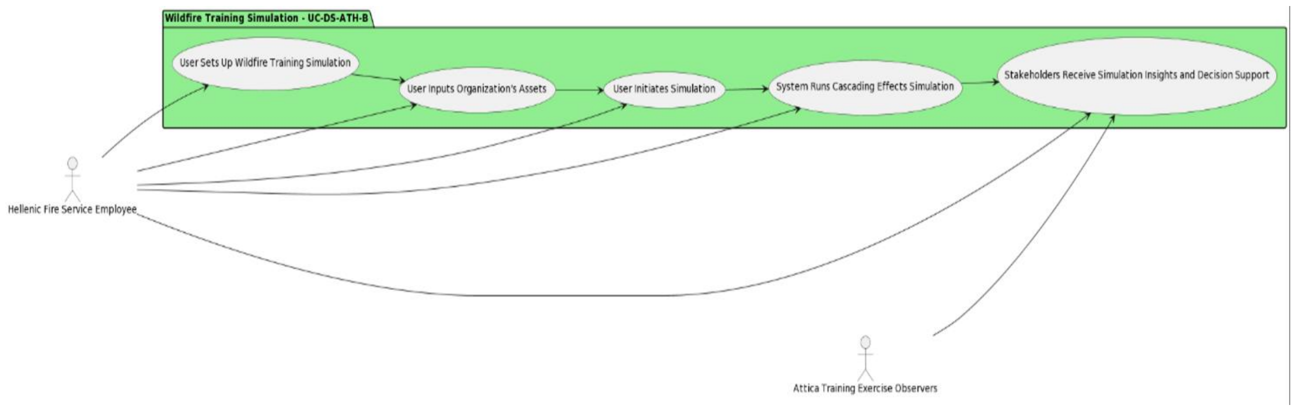

Wildfire Training Simulation - UC-DS-ATH-B
User Sets Up Wildfire Training Simulation
User Inputs Organization's Assets
User Initiates Simulation
System Runs Cascading Effects Simulation
Stakeholders Receive Simulation Insights and Decision Support
Hellenic Fire Service Employee
Attica Training Exercise Observers

Figure 4 Wildfire simulation use-cases (as in D3.6)

The way the statistical analysis of data and statistical simulations are applied is described below.

UC-DS-ATH-B-1-**User Sets Up Wildfire Training Simulation:**

This initial use case involves a user (specifically a Hellenic Fire Service Employee) configuring the wildfire simulation environment. The setup includes defining simulation parameters such as location, severity, expected weather conditions, and other relevant inputs to create a realistic training scenario.

The simulation parameters are obtained by analysing existing data, and by selecting the simulation type.

UC-DS-ATH-B-2-**User Inputs Organization's Assets:**

The user then provides details about the assets available for the simulation. These assets could include firefighting vehicles, personnel, equipment, and any infrastructure the organization intends to use in response to the wildfire. This step ensures that the simulation accurately reflects the organization's capacity to respond to a wildfire.

UC-DS-ATH-B-3-**User Initiates Simulation:**

After configuring the setup and inputting organizational assets, the user starts the simulation. This action triggers the system to begin the simulated wildfire scenario, putting the predefined assets and conditions into action. It can select one of the statistical simulations presented for this scenario. This use case uses the analysis of statistical data done for UC-DS-ATH-B-1.

System Runs Cascading Effects Simulation:

Once the simulation is initiated, the system processes the wildfire scenario, taking into account factors such as wind, terrain, and vegetation. This "cascading effects" component likely refers to how the fire spreads and interacts with the environment, impacting nearby assets and possibly creating secondary events (such as smoke impacting nearby towns or fire-threatening infrastructure). This use case (the simulation) is based on the parameters entered in previous use cases. To define the right parameters the user makes use of the statistical analysis of data, then of the results of statistical simulations.

Stakeholders Receive Simulation Insights and Decision Support:

The final use case involves stakeholders, including Attica Training Exercise Observers and other relevant authorities, receiving insights from the simulation. These insights might include data on wildfire spread, asset utilization, and decision support for handling similar real-life incidents. This output provides critical information to guide training, improve decision-making, and refine response strategies. This use case is indirectly affected by the statistical analysis of existing data, as this is an input for running the simulations.

3.3.3 SCENARIO

1. **KPIs**: Define Key Performance Indicators and Thresholds for Fire Prediction Accuracy
2. **Initial Monitoring:** The model continuously receives data on current temperature, wind speed, humidity, and vegetation type.
3. **Detecting Need for Adaptation:** If wind the speed suddenly increases, the model's anomaly detection system flags this as a potential risk factor. It consults its decision rules and decides that adaptation is necessary to maintain prediction accuracy.
4. **Decision and Adaptation:** Based on reinforcement learning, the model has learned that increased wind speed correlates with faster fire spread. It adjusts its wind influence parameter, recalculates fire spread rates, and updates the simulation.
5. **Feedback and Learning:** After the adaptation, the model receives real-time satellite data confirming its updated predictions closely match the observed fire spread. It logs this outcome, reinforcing the adaptation strategy.
6. **Predictive Proactive Adaptation:** Later, a weather forecast predicts low humidity in the next few hours. The model pre-emptively adjusts vegetation moisture levels to simulate how the fire might behave under drier conditions, helping firefighting teams prepare.

By applying predictive/proactive in this way, wildfire simulation models can dynamically adjust to real-world changes, improving their ability to provide timely and accurate predictions that support firefighting and risk assessment efforts. This capability ultimately enhances the model’s value as a decision-support tool in high-stakes, rapidly evolving wildfire scenarios.

3.3.4 USER INTERFACE AND SERVICES OFFERED

The simulation is displayed on a local canvas or saved as a PNG file, which can be opened in other applications. Additionally, the simulation can be executed via a REST API, producing either a PNG file depicting the fire representation or a JSON file containing the final computed parameters after applying the adaptive algorithm.

The best results are obtained when running in interactive mode, and the user can conduct the way the simulation is done.

An example of the output image is presented in the next figure (Figure 5 Wildfire simulation):

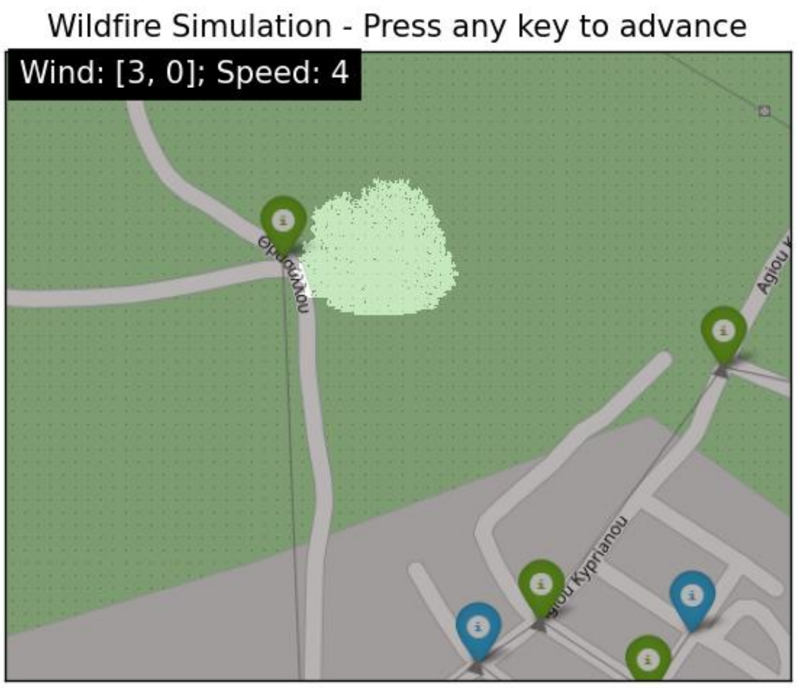


Figure 5 Wildfire simulation

To run REST services, the following APIs are available:

Initialize fire

Send a POST request to `<URL>/initialize-fire` with a JSON body like:

```
{
    "grid": [[0, 1, 0], [1, 0, 1], [0, 1, 0]],
    "center": {"lat": 38.107983, "long": 23.66651},
    "radius": 5,
    "method": 2
}
```

The expected result is similar with:

```
{
    "status": "success",
    "data": {
        "message": "Fire initialized",
        "grid_size": [3, 3],
        "center": {"lat": 38.107983, "long": 23.66651},
        "radius": 5,
        "method": 2
    }
}
```

Spreads fire

Send a POST request to `<URL>/spread-fire` with JSON body:

```
{
    "grid": [[0, 1, 0], [1, 0, 1], [0, 1, 0]],
    "fire": [[0, 0, 0], [0, 1, 0], [0, 0, 0]],
    "wind": [1, -1],
    "prob_spread": 0.4,
    "speed": 2
}
```

The expected results are similar with:

```
{
    "status": "success",
    "data": {
        "message": "Fire spread simulated",
        "grid_size": [3, 3],
        "fire_size": [3, 3],
        "wind": [1, -1],
        "prob_spread": 0.4,
        "speed": 2
    }
}
```

Adjust parameters

Send a POST request to <URL>`/adjust-parameters` with JSON body:

```
{
    "grid": [[0, 1, 0], [1, 0, 1], [0, 1, 0]],
    "fire": [[3, 0, 2], [2, 1, 0], [4, 0, 0]],
    "wind": [1, -1],
    "prob_spread": 0.4,
    "speed": 2
}
```

The expected result is similar with:

```
{
    "status": "success",
    "data": {
        "message": "Parameters adjusted",
        "grid": [[0, 1, 0], [1, 0, 1], [0, 1, 0]],
    `   "fire": [[2, 0, 1], [1, 1, 1], [2, 0, 1]],
        "wind": [1, -1]

    }
}
```

Note: when running the server locally the URL could be: `http://127.0.0.1:8000`

3.3.5 INTERPRETATION OF THE DATA, AND RESULTS

The wildfire simulation utilizes diverse datasets to model fire behaviour, assess risk, and optimize response strategies. These datasets align with the traffic adaptation model to ensure a seamless and effective disaster response:

- The system integrates vegetation maps to predict fire spread, combining this with road network data for optimal evacuation routing. This data is specified when initial parameters are set.

- Terrain data is used alongside traffic models to identify high-risk zones and prioritize road closures and safe routes. This data is specified when initial parameters are set.
- Wind speed and direction influence fire dynamics. The AI system dynamically adapts road usage, intersection capacities, and evacuation routes in response to changing weather conditions. The data is entered interactively during the simulation run.
- The traffic simulation integrates this data to facilitate rapid response and ensure firefighting teams reach critical areas efficiently. It is done by generating traffic based on the statistical distributions chosen.

To support firefighting operations and public safety, traffic adaptation plays a crucial role:

- The system leverages OpenStreetMap data and queue theory-based congestion analysis to implement curfew measures for fire response teams.
- Routes are dynamically adjusted using AI-based adaptation, including rerouting and road closures.
- The system ensures that evacuation paths avoid areas with high fire risk while minimizing congestion at key intersections.
- The AI system integrates resource allocation with traffic control measures to prevent bottlenecks.
- Locations of hospitals, power plants, and water sources are factored into response planning.
- Adaptive algorithms ensure these areas remain accessible while optimizing traffic and firefighting routes.

To enhance real-time decision-making, the system provides:

- **Visualization:** Displays results using charts, graphs, and maps to enhance situational awareness.
- **REST API Exposure:** All services, including traffic adaptation, fire spread predictions, and evacuation route computations, are exposed via APIs for seamless integration with emergency response systems.

3.4 SELF-ADAPTIVE EARTHQUAKE MODELS

Our approach presents an AI-enabled auto-adaptive traffic simulation that models human behaviour and vehicle movement in the situation of an earthquake, using OpenStreetMap[3] data. The system creates a graph representation of the road network, dynamically computes shortest paths, analyses congestion, and applies adaptive AI strategies, including rule-based and reinforcement learning (RL) techniques, to optimize traffic flow.

3.4.1 KEY FACTORS IN EARTHQUAKE SIMULATION

The key factors used by this auto-adaptive AI-enabled traffic simulation include:

Traffic Network Representation

The road network is extracted from OpenStreetMap and represented as a graph:

- **Nodes**: Represent intersections with defined capacities, indicating how many vehicles can pass per unit of time.
- **Edges**: Represent roads with length attributes determining travel distances.

[3] https://www.openstreetmap.org

Two key interest points are selected to simulate human movement and traffic flow. The graph is refined by removing cycles and computing the shortest paths between the two locations. The routes are adjusted dynamically when intersections or roads become unavailable due to traffic or disasters.

To mitigate single points of failure, the system computes independent shortest paths, ensuring that critical intersections do not appear in multiple routes. These paths are displayed for user analysis.

Traffic Flow Simulation Using Queue Theory (Lee, 1966)

The traffic flow along the shortest and independent paths is simulated using an M/M/c queue model. Different statistical distributions (Poisson, Pareto, Weibull, and Exponential) are used to introduce stochastic variations in traffic conditions.

Key outputs include:

- **Queue Length Computation**: Indicates congestion levels at intersections.
- **Visualization**: Queue lengths are presented in charts and overlaid on maps for user interpretation.
- **Path Closure Simulation**: The system assesses the impact of closed intersections by dynamically recomputing new independent paths in real-time.

Auto-Adaptive AI Optimization

Adaptive AI strategies are implemented at multiple stages to dynamically adjust traffic flow and optimize responses in disaster scenarios.

- **Path Recalculation Under Disruptions:** The system simulates intersection closures and immediately searches for new independent paths to ensure continued traffic movement.
- **Rule-Based Adaptation:** Traffic signal timings and lane allocations are dynamically adjusted to reduce congestion. This also is done to minimize queue lengths.
- **Reinforcement Learning (RL) Adaptation**: An RL model optimizes intersection capacities to minimize queue lengths over successive iterations. It is based on Q-Learning (Lapan, 2020)
- **Saving Adaptive Parameters:** The final optimized capacities of intersections and roads are stored for future reference. Results, including node capacities and edge lengths, are exported to CSV files for further analysis.
- **Failure Notification**: If no viable solution exists under certain traffic conditions, the system notifies the user, enabling contingency planning.

Data Storage & Analysis

- **CSV Data Export:** Saves node capacities, edge lengths, and queue lengths for future analysis.
- **Failure Detection:** Notifies users if no feasible routes exist, allowing for contingency planning.

3.4.2 USE CASES

The use cases specific to the earthquake scenario are presented in the next figure extracted from D3.6.

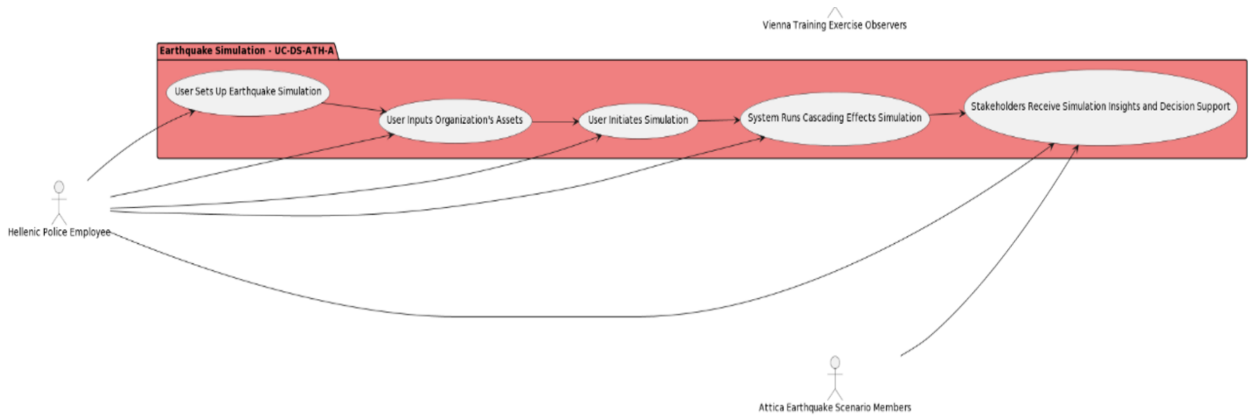

Vienna Training Exercise Observers
Earthquake Simulation - UC-DS-ATH-A
User Sets Up Earthquake Simulation
User Inputs Organization's Assets
User Initiates Simulation
System Runs Cascading Effects Simulation
Stakeholders Receive Simulation Insights and Decision Support
Hellenic Police Employee
Attica Earthquake Scenario Members

Figure 6 Earthquake simulation use cases (as in D3.6)

This diagram describes an earthquake preparedness simulation, allowing users to configure a realistic earthquake scenario, initiate the simulation, and analyse its cascading effects. Stakeholders receive detailed insights and decision support to improve their response strategies and enhance readiness for actual earthquake incidents. This setup provides a comprehensive tool for training, decision-making, and evaluating resource allocation in earthquake scenarios.

UC-DS-ATH-A-1-User Sets Up Earthquake Simulation:

In this step, a user, specifically a Hellenic Police Employee, configures the earthquake simulation. The setup would include defining parameters relevant to an earthquake scenario, such as magnitude, epicentre location, potential aftershocks, and affected infrastructure.

UC-DS-ATH-A-2-User Inputs Organization's Assets:

The user provides details about the resources and assets available to respond to the earthquake, such as police units, emergency personnel, vehicles, and medical supplies. These inputs help the system simulate a realistic response scenario based on the organization's available assets. This use case is not affected by the statistical analysis of existing data.

UC-DS-ATH-A-3-User Initiates Simulation:

Once the earthquake scenario is set up and assets are input, the user initiates the simulation, triggering the system to start the simulated earthquake event. The statistical-based simulations are run. They will use the parameters selected in a previous use case, and the algorithms specific to the target values simulated. (see next subchapters). The statistical analysis of data gives input to this step, considering the parameters of the statistical models.

UC-DS-ATH-A-4-System Runs Cascading Effects Simulation:

The system processes the earthquake scenario, simulating the cascading effects of the earthquake. This may include aftershocks, structural damage, potential fires, or other secondary crises that could arise following an earthquake. The cascading effects simulate the broader impact on infrastructure, populations, and available assets. This use case is indirectly affected by the statistical analysis of existing data, as the inputs may consider the statistical parameters or the result of simulations using statistical models.

UC-DS-ATH-A-5-Stakeholders Receive Simulation Insights and Decision Support:

The final step involves stakeholders, including Vienna Training Exercise Observers and Attica Earthquake Scenario Members, receiving insights and decision support from the simulation. These insights may cover the impact on affected areas, asset deployment efficiency, and recommendations for handling similar real-world incidents. This information is critical for improving preparedness, refining response plans, and enhancing training for future earthquake events. This use case is indirectly affected by the statistical analysis of existing data, considering that the statistical analysis gives inputs for the whole simulation process.

3.4.3 SCENARIO

1. **KPIs**: The main target of the application of the algorithms is the establishment of the parameters (nodes, edges with their capacity, so that:
    a. The paths between two points exist if previously it did not exist.
    b. The path length is the smallest possible. The KPI is the report between the length of the shortest path and the medium length of the paths.
    c. The queue length after applying the self-adaption is smaller than the initial one. The KPI is the report between the maximum queue length after applying the algorithm per maximum queue length before applying the algorithms.
2. **Initial Monitoring:** The traffic simulation continuously gathers real-time congestion data from sensors, mirroring how seismic models receive ground-shaking data from fault-line sensors.
3. **Detecting Need for Adaptation**: Similar to seismic anomaly detection, traffic anomalies (e.g., sudden congestion spikes due to an accident or road blockage) trigger adaptive measures in the traffic model.
4. **Decision and Adaptation**: Just as seismic models recalibrate soil amplification parameters, the traffic model dynamically modifies intersection capacities and reroutes traffic to optimize flow in real-time.
5. **Proactive Aftershock Adaptation**: Post-disaster, the system simulates weakened infrastructure, enabling a more accurate risk assessment of future congestion or road failures.
6. **Feedback and Learning**: After each adaptation, the AI model compares its predictions with actual traffic flow data, refining parameters for future scenarios, much like how seismic models refine soil and structural response data post-event.

3.4.4 AI METHODS FOR EARTHQUAKE MODEL ADAPTATION

The earthquake simulation is grounded in computational graph theory, queueing models, and reinforcement learning to analyse and adapt traffic movement in seismic events. The approach is structured as follows:

Road Network Representation

- Data is collected from OpenStreetMap, where intersections are modelled as nodes and roads as edges in a graph representation.
- The graph structure allows pathfinding and network analysis for effective traffic simulations during earthquakes.

Path Computation and Adaptation (Berge, 1982)

- The shortest paths between key locations are computed to simulate human movement and evacuation routes (Berge, 1982).
- Independent shortest paths are identified to prevent reliance on common intersections, reducing the risk of single points of failure.

- The system dynamically recomputes paths when intersections or roads become unavailable due to earthquake damage.

Traffic Simulation with Queue Theory (Lee, 1966)

- Traffic congestion is modelled using M/M/c queueing systems, where intersections act as multi-server queues.
- Queue lengths at intersections are computed to evaluate congestion levels and bottlenecks.
- Statistical distributions (Poisson, Weibull, Pareto, Exponential) simulate stochastic variations in traffic flow, capturing real-world uncertainties.

Auto-Adaptive Traffic Control

- Rule-Based Adaptation: Intersection capacities are adjusted dynamically using predefined heuristics to reduce queue lengths.
- Reinforcement Learning (RL) (Lapan, 2020): AI learns optimal traffic management strategies by continuously adjusting intersection and road capacities based on observed traffic patterns.
- Adaptive controls ensure that congested roads and intersections dynamically evolve based on real-time conditions.

Data Storage and Analysis

- Results, including nodes (intersection capacities) and edges (road lengths), are stored in CSV format for future simulations and analysis.
- If no feasible evacuation route is found, the system notifies users, enabling proactive disaster response planning.

Output Representation and API Integration

- **Visualization:** Results are displayed through **charts, graphs, and maps**, providing intuitive insights into congestion, queue lengths, and alternative routes.
- **REST API Exposure:** All functionalities, including path computation, queue length analysis, and adaptive traffic control, are exposed as **RESTful APIs**, enabling seamless integration with external disaster management and urban planning systems.

This computational framework ensures a real-time, adaptive, and data-driven approach to earthquake-related traffic management, improving urban mobility under seismic stress conditions.

This AI-driven, auto-adaptive traffic simulation provides a robust framework for optimizing vehicle movement in disaster scenarios. By leveraging real-time path recalculation, queue theory-based congestion analysis, AI-driven adaptive learning, and comprehensive API integration, the system enhances urban resilience and emergency response strategies. Future work will integrate additional real-time sensor inputs, satellite imagery, and advanced predictive modelling for improved disaster mitigation.

The details of this approach are presented in the next subchapters.

3.4.5 EARTHQUAKE SIMULATION MODEL USAGE DETAILS

3.4.5.1 *Presentation in the user interface*

The simulation is displayed on a local canvas or saved as a PNG file, which can be opened in other applications when is about graphs and charts. Information on the map is displayed as HTML pages which are also saved locally for further reference. Additionally, the simulation can be executed via a REST API, producing either a

PNG file depicting the fire representation, an HTML file representing information on a map, or a JSON file containing the final computed parameters after applying the adaptive algorithm.

The user interface and REST API presented will follow the main steps of running the simulation.

3.4.5.2 *Road Network Representation.*

This is the first step in our simulation model. We are building here the graph used to model the streets and the intersections.

3.4.5.2.1. Inputs

OpenStreetMap[4] system.
Data is read using the osmnx[5] Python library. The call is:

```
# Retrieve the street network
G = ox.graph_from_place(place_name, network_type="all")
```

place_name is the name of a place recognized by OpenStreetMap. Example used:

```
# PLACE_NAME = "Fili, Greece"
# PLACE_NAME = "Attica, Greece"
```

3.4.5.2.2. Outputs

csv files, ttl files to be used in GraphDB or neo4j.

Data is read from OpenStreetMap and saved locally in csv files as nodes and as edges.

An example of a nodes file and an edges file is below:

Nodes file

```
osmid,y,x
26733684,37.9169119,23.7021355
26734032,37.9163994,23.7053716
26734041,37.9168317,23.70687
```

Edges file

```
u,v, name, length
26733684,7326480515,Unnamed Road,55.138041356770685
26733684,9018040029,Unnamed Road,91.7310126681424
26734032,4015873338,Ποσειδώνος,192.84253930809896
```

The nodes file and edges file are also converted in TTL format to be placed in the GraphDB database. An example of ttl file is below:

```
    @prefix : <http://example.org/roadnetwork#> .
    @prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
    @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
    @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
```

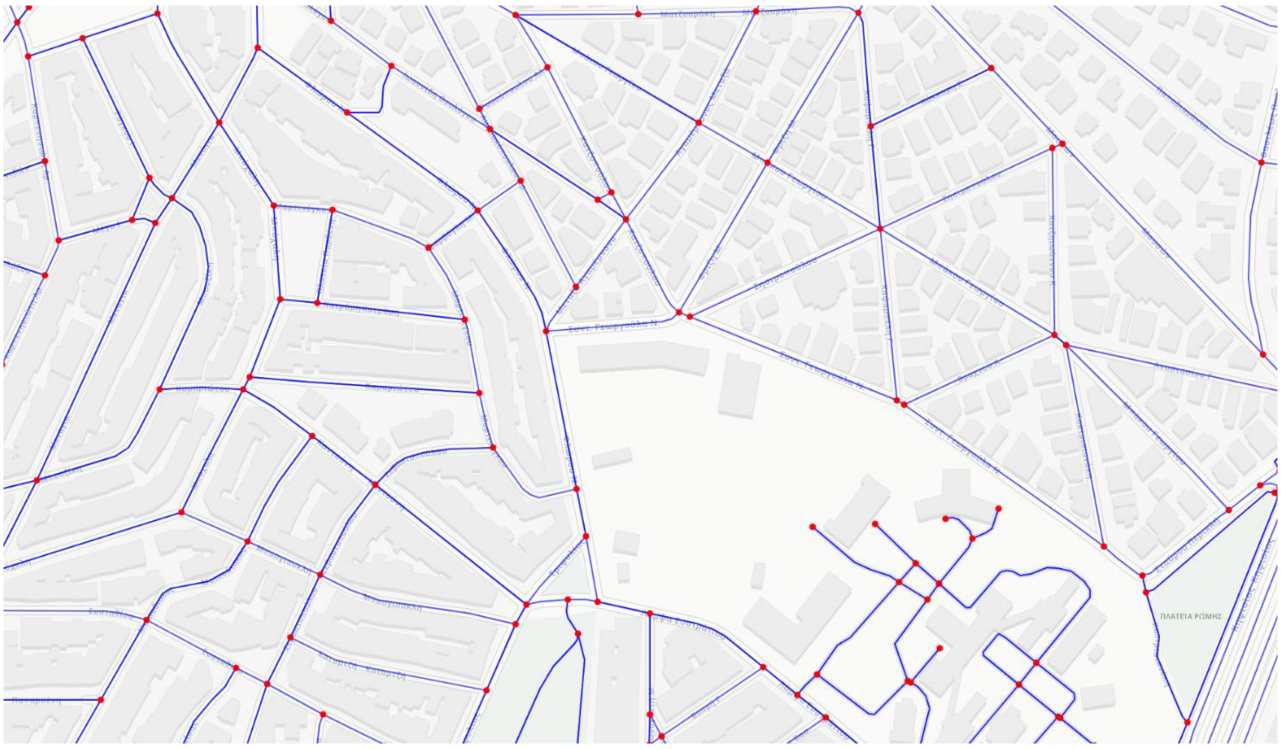
[4] https://www.openstreetmap.org/
[5] https://osmnx.readthedocs.io/en/stable/user-reference.html

```
:Node12368717021 :connectsTo :Node932024473 ;
            :roadName "Θρασύλλου" ;
            :length "7.156062669799672"^^xsd:float .
:Node12368717021 :connectsTo :Node11266622819 ;
            :roadName "Θρασύλλου" ;
            :length "58.88768692868626"^^xsd:float .
:Node12368717021 :connectsTo :Node11266622823 ;
            :roadName "Unnamed Road" ;
            :length "68.96437329715064"^^xsd:float .
```

The map with the nodes and roads is then represented in an HTML file, like in the next figure: (Figure 7 Road network representation)
Here we have a map, the roads are represented by blue lines and the intersection by red circles.
For further process, we are interested in the Grapg which is constructed. The graph representation is in fact what is visible in the next figures, but without the background map. Only the blue lines representing graph edges and the red points representing nodes are considered by the graph representation. The weight of an edge is the length of the road between two intersections.

3.4.5.2.3. User Interface

The created graphs are represented on the map. An HTML file is created, and by opening the html file in a browser, the whole OpenStreetMap functionality, plus the representation of the graphs is displayed. This is presented in the next figures.
At this stage the visual output displays the graph, indicating how complex it is.
The output is the graph definition in terms of nodes and edges and is saved in csv files to be used in the next steps. Once the data is retrieved from OpenStreetMap, the graph is constructed and saved in csv files. For the next steps, it is possible to edit the csv files with nodes and edges and work on the edited files.
That means that we can restrict the graph to only the nodes and edges of interest and remove the full complexity of the graph read from OpenStreetMap.
In our particular example from a graph with some thousands of nodes, we reduce it to about 100 nodes of interest. This way the whole computations are very fast.



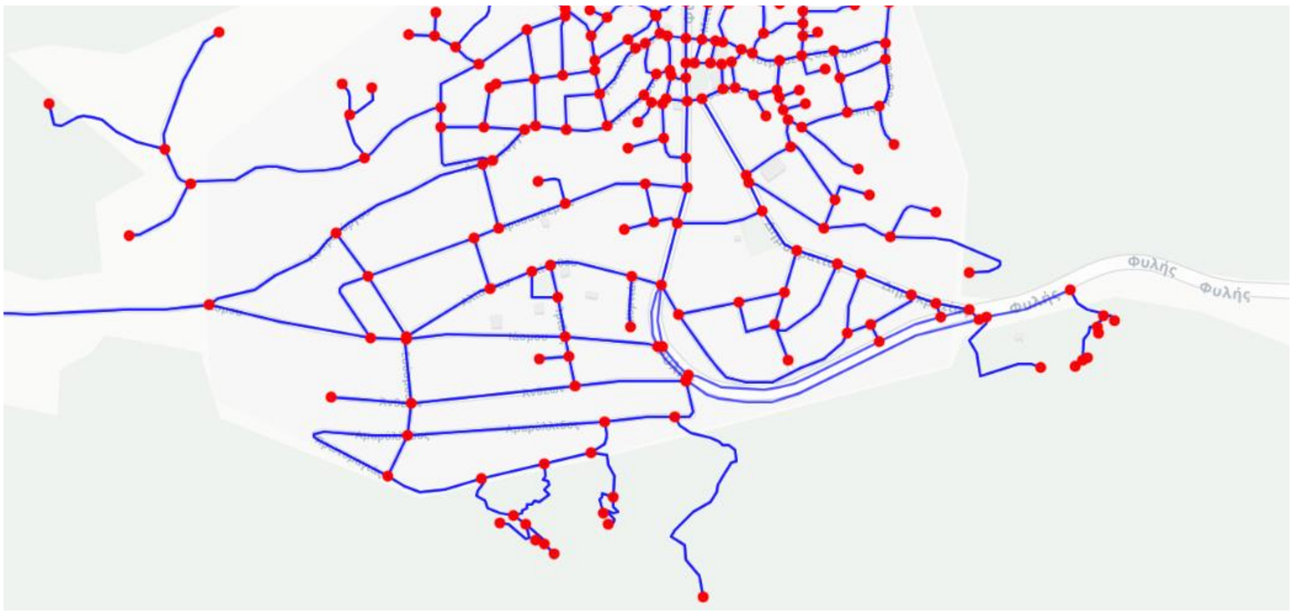


Figure 7 Road network representation

3.4.5.2.4. The REST APIs

Read data from open street

Send a POST request <URL>/`read-openstreet` with a JSON body:

```
{ "place_name": "Fili, Greece" }
```

Example response:

```
{
    "status": "success",
    "place": "Fili, Greece",
    "nodes": "{... GeoJSON Data ...}",
    "edges": "{... GeoJSON Data ...}"
}
```

Read data from the OpenStreetMap, save nodes and edges locally and return the file names.

Send a POST request to `<URL>/read-openstreet-save` with a JSON body:

```
{
    "place_name": "Fili, Greece",
    "nodes_file": "nodes_fili.csv",
    "street_map_file": "edges_fili.csv"
}

Example Response
{
    "status": "success"
    "nodesFileName": "nodes_fili.csv",
    "edgesFileName": "edges_fili.csv"
}
```

Read data from OpenStreetMap, create an HTML map, and return the name of the HTML.

Send a POST request to `<URL>/generate-map` with a JSON body:

```
{
    "place_name": "Fili, Greece"
}
```

Example response

```
{
    "status": "success",
    "htmlFileName": "Fili.html"
}
```

3.4.5.3 *Path Computation and Adaptation*

This is the second step and allows us to compute the paths between two points of interest.
After reading the data from OpenStreetMap for a specific area, we have to select two points: Start point and End Point. Then we will compute the paths between the two points.
The cycles are removed. The shortest paths are computed, and finally, the independent paths are computed.

3.4.5.3.1. Inputs:

1. Graph representation data obtained in the previous step. This should be in two csv files. One describes the nodes and one describes the edges. Example of such data:

Nodes file, a csv file with the content similar to:

```
osmid,y,x
26733684,37.9169119,23.7021355
26734032,37.9163994,23.7053716
26734041,37.9168317,23.70687
```

Edges file, a csv file with the content similar to:

```
u,v, name, length
26733684,7326480515,Unnamed Road,55.138041356770685
26733684,9018040029,Unnamed Road,91.7310126681424
26734032,4015873338,Ποσειδώνος,192.84253930809896
```

2. Start and end node.
   Example: `26733684, 26734032`

3.4.5.3.2. Processing steps and methods:

The steps considered are:

1. *Cycle removal*
2. *All (shortest) path computation*
3. *Independent paths selection*

1. *Cycle removal*.
   Like all other graph algorithms, the Python networkx library is used. Particularly for the cycle removal, the call of networkx[6] procedures is done in the following code:

```
while True:
    try:
        # Attempt a topological sort
        cycle=nx.find_cycle(graph, orientation='original')
       # If a cycle exists, remove one edge from the cycle

       graph.remove_edge(*cycle[-1][:2])

    except nx.NetworkXNoCycle:
        # No cycles remain
        break
```

`find_cycle` method is implemented in networkx using a cycle search via depth-first traversal.

2. *Find the paths between the two nodes in increasing total length*

After removing the cycles, the paths between the start and end node are computed. Input is the graph resulting after cycle removal.

The Python code, calling networkx graph procedures is:

```
all_paths = list(nx.all_simple_paths(graph, source=source, target=target))
# a better variant is to use a method based on the Djikstra algorithm
# all_paths = list(nx.all_shortest_paths(graph, source=source, target=tar-
get))

path_lengths = [(path, sum(graph[u][v]['length'] for u, v in zip(path[:-1],
path[1:]))) for path in all_paths]
sorted_paths = sorted(path_lengths, key=lambda x: x[1])
```

The algorithm used by the all_simple_paths method is a modified depth-first search to generate the paths[7].

The algorithm used by the all_shortest_paths method is a modified Dijkstra's algorithm to search the paths[8].

When the graph is simple (hundred of nodes) `all_simple_paths` method is used as it is fast and gives all the paths.

When the graph becomes more complex then it is recommended first to test if there exist paths between the start and end node, the run several times `all_shortest_path` to get the required paths.

[6] https://networkx.org/

[7] https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.simple_paths.all_simple_paths.html

[8] https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.shortest_paths.generic.all_shortest_paths.html#networkx.algorithms.shortest_paths.generic.all_shortest_paths

3. *Find independent paths between the two nodes.*

Having as input all the paths detected between the start and end node, in the next step only the independent paths are computed.

The code used is:

```
# Filter independent paths
independent_paths = []
used_nodes = set()
for path, length in sorted_paths:
if not any(node in used_nodes for node in path[1:-1]):
        independent_paths.append(path)
        used_nodes.update(path[1:-1])  # Avoid reusing intermediate nodes

return independent_paths
```

If no paths are available, notifications are displayed.

3.4.5.3.3. Outputs:

- Html file, containing the maps and the graph representation
- A text file enumerating the paths

In the figure: Figure 8 Paths between two points on the map detected paths are represented in magenta colour on the map and graph.

In the figure: Figure 9 Independent Paths between two points on the map, the independent paths are displayed in magenta colour.

If we compare figures 7 and 8, we can see that in figure 8 there are not nodes present on the two paths (except start and end)

The results are displayed on the map and saved as html format.

The representation uses lines between points not following the streets for a better understanding of the representation.

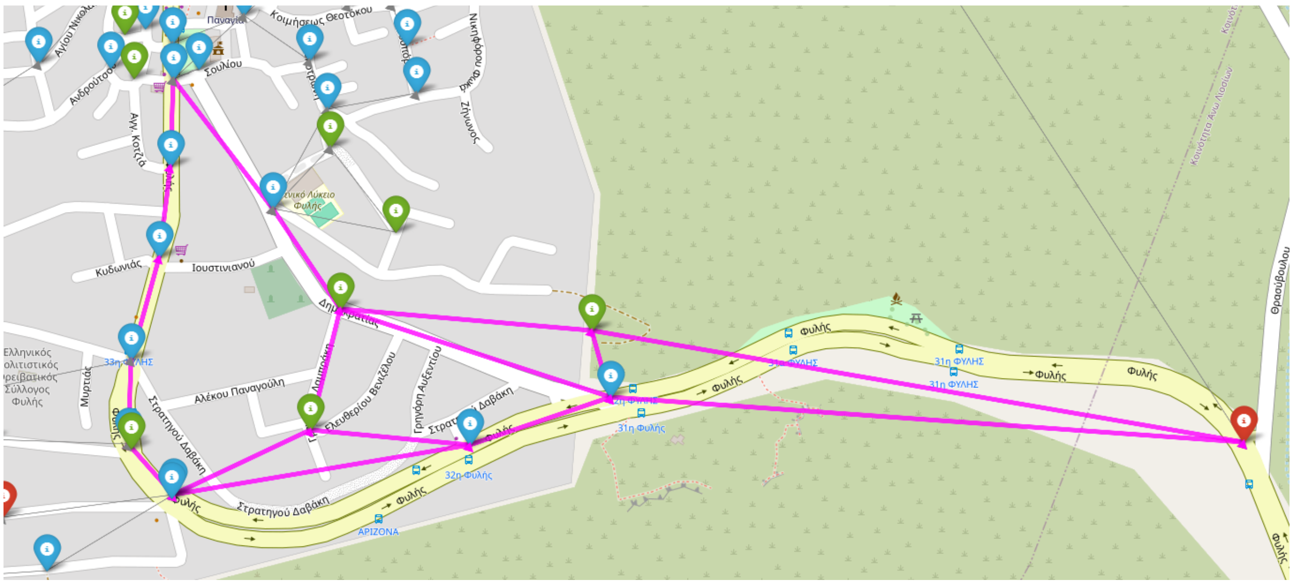


Figure 8 Paths between two points on the map

Now the independent Paths are represented in the next figure (Figure 9 Independent Paths between two points on the map):

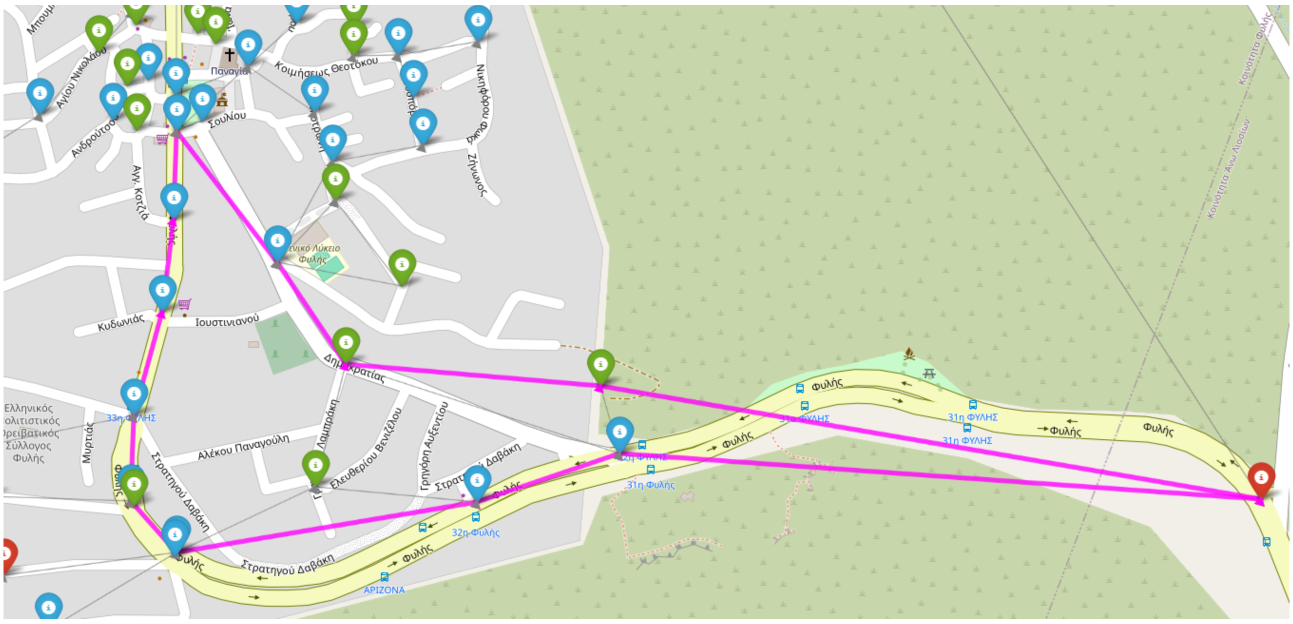


Figure 9 Independent Paths between two points on the map

The output is useful for the user to visualize the paths, and for the system to have the paths saved for the next steps.

3.4.5.3.4. REST APIs:

Besides the graphical representation of paths on a map, it is also possible to call REST APIs which make the same computations and return the paths as a list of nodes

The REST APIs developed are

```
@app.post("/compute-shortest-path")
async def api_compute_shortest_path(request: PathRequest):


@app.post("/compute-independent-path")
async def api_compute_independent_path(request: PathRequest):
```

3.4.5.4 *Traffic Simulation with Queue Theory*

Now, after the previous step, having the paths and the independent paths we can simulate the traffic by using queue theory. We employ an M/M/c queue model and generate "cars" going in different intersections. Each intersection has a capacity (the number of cars possible to pass the intersection in a time interval). After running a simulation, we obtain a description of queue lengths (cumulative) and a graph presentation like in the next figure (Figure 10 Queue length variation during trials):

The details of this simulation are presented in the next paragraphs.

3.4.5.4.1. Inputs

1. Graph representation data obtained in the previous step. This should be in two csv files. One describing the nodes and one describing the edges. Example of such data:

Nodes file, a csv file with the content similar to:

```
osmid,y,x
26733684,37.9169119,23.7021355
26734032,37.9163994,23.7053716
26734041,37.9168317,23.70687
```

Edges file, a csv file with the content similar to:

```
u,v, name, length
26733684,7326480515,Unnamed Road,55.138041356770685
26733684,9018040029,Unnamed Road,91.7310126681424
26734032,4015873338,Ποσειδώνος,192.84253930809896
```

2. Start and end node.
   Example: `26733684, 26734032`
3. Simulation parameters (They define the type of algorithm used as described in the next paragraphs):

```
a. NO_TRIALS = 200           # Number of simulations run
b. ARRIVAL_LAMBDA = 50       # Requests per time unit
c. MEAN_SERVICE_TIME = 20     # average number of requests served in a
   time unit
d. QUEUE_CAPACITY = 20       # Maximum capacity of the queue
```

These are inputs also for the previous step. For this particular step, the output of the previous step i.e. the Paths detected are inputs.

3.4.5.4.2. Models and algorithms

The data is processed by applying queue simulation of type M/M/c and M/M/c/K

The notation M/M/c follows Kendall's notation [(Lee, 1966)], where:

- M (Markovian arrivals) = The inter-arrival times follow an exponential (Poisson) process with rate λ (ARRIVAL_LAMBDA input parameter)
- M (Markovian service times) = The service times are exponentially distributed with rate μ per server (MEAN_SERVICE_TIME input parameter).
- c (Number of servers) = There are ccc servers in parallel, all serving customers from a single queue (Tested for 1 and 2).
- K (Queue capacity) = Maximum capacity of a queue (QUEUE_CAPACITY input parameter)

3.4.5.4.3. Performance metrics

For the simulation considered, we can compute the following performance metrics:

Expected Number of Requests in System (Little's Law)

Using **Little's Law** (L=λW), the average number of requests in the entire system (queue + service) is:

$$L = L_q + \frac{\lambda}{\mu}$$

Where:

- L: Average number of requests in the system (including those being served)
- L_q: Average number of requests in the queue (waiting)
- λ: Arrival rate (e.g. request per unit time)
- μ: Service rate (e.g. requests served per unit time)

Expected Number of Requests in Queue

The expected number of requests waiting in the queue **is:**

$$L_q = \frac{P_w \cdot \rho \cdot c}{1 - \rho_c}$$

Where:

- L_q: (Expected Number of requests in Queue)

This represents the average number of requests waiting in the queue before service (excluding those currently being served).

It is a key performance metric to measure congestion in the system.

- P_w :(Probability That an Arriving Request Has to Wait)

This is the probability that all c servers are busy, meaning an arriving request must wait in the queue before being served.

- ρ: Traffic intensity (usualy $\rho = \frac{\lambda}{c\mu}$)
- c: Number of servers
- ρ_c: Utilization factor per server

It is computed using the Erlang C [(Lee, 1966)] formula:

$$P_w = \frac{(\lambda/\mu)^c}{c!} \cdot \frac{c\mu}{c\mu - \lambda} P_0$$

Where P_0 is the probability that there are zero requests in the system and μ is the service rate:

$$P_0 = \left[\sum_{n=0}^{c-1} \frac{(\lambda/\mu)^n}{n!} + \frac{(\lambda/\mu)^c}{c!} \cdot \frac{c\mu}{c\mu - \lambda} \right]^{-1}$$

- ρ (Traffic Intensity per Server)

It is the ratio of arrival rate to service rate per server:

$$\rho = \frac{\lambda}{\mu}$$

This represents the fraction of time that each server is busy, also known as server utilization.

- c (Number of Servers)

This is the total number of servers available in the system.

It affects the queue length because having more servers reduces waiting times.

- ρ_c (Overall System Utilization)

The system utilization is given by:

$$\rho = \frac{\lambda}{c\mu}$$

This represents the fraction of the time that the system (all servers combined) is busy.

When ρ_c approaches 1 (i.e., the system is highly utilized), waiting times increase significantly.

- $1-\rho_c$ (Idle Time Factor)

This represents the probability that at least one server is idle at any given moment.

It acts as a denominator to normalize the equation and prevent infinite values when the system is heavily loaded.

Average Waiting Time in Queue

The average **waiting time in the queue** (before being served) is:

$$W_q = \frac{L_q}{\lambda}$$

The components of the formula are explained in the previous parameter description.

Average Time Spent in the System

The total time a customer spends in the system (waiting + service time) is:

$$W = W_q + \frac{1}{\mu}$$

For example, on the first try, we have:

Inputs

Simulation time: 1000 minutes

Number of servers: 4

Arrival rate: 50 calls/minute

Service rate: 20 calls/minute per server

Outputs

Average waiting time in queue: 0.0115 minutes

Maximum waiting time in queue: 0.3121 minutes

Average time in system (waiting + service): 0.0617 minutes

Approximate server utilization: 0.6281

In a situation when the capacity is half like in the previous one, we have:

Inputs

Simulation time: 1000 minutes

Number of servers: 4

Arrival rate: 50 calls/minute

Service rate: 10 calls/minute per server

Outputs

Average waiting time in queue: **102.4276 minutes**

Maximum waiting time in queue: **200.8298 minutes**

Average time in system (waiting + service): **102.5179 minutes**

Approximate server utilization: **0.9996**

3.4.5.4.4. Simulations

Now the simulation randomly generates requests in all nodes of the given path in the graph. The paths used are the output of the previous step where the independent paths were computed between two nodes.

A discrete-event simulation using the M/M/c model tailored for this scenario. In this example, calls arrive following a Poisson process, and call durations (service times) are exponentially distributed. We use a multi-server resource (representing call agents) to process calls. You can adjust parameters like the arrival rate, service rate, and number of servers to fit a specific use case.

The **Poisson** distribution is described as:

The probability of observing k events in a given interval (where k is a non-negative integer, 0, 1, 2, 3, ...) is given by the formula:

$$P(k;\lambda) = \frac{\lambda^k e^{-\lambda}}{k!}$$

Where:

- $P(k;\lambda)$ is the probability of k events occurring in the fixed interval.
- λ is the **average rate of occurrence** (the expected number of events in the interval).
- e is Euler's number, approximately 2.71828.
- k! is the factorial of k (i.e., k×(k−1)×(k−2)×⋯×1).

The **exponential** distribution is described as:

The probability density function (PDF) of the exponential distribution is given by:

$$f(x;\lambda) = \lambda e^{-\lambda x}, \quad x \geq 0$$

Where:

- f(x;λ) is the probability density function (the likelihood of observing a value close to xxx).
- λ is the **rate parameter**, representing the average rate at which events occur.
- **Note**: λ is also the inverse of the mean (λ=1/μ), where μ is the average time between events.
- e is Euler's number, approximately 2.71828.
- x≥0x since the time or distance between events cannot be negative.

The **cumulative distribution function (CDF)**, which gives the probability that a random variable X is less than or equal to x, is:

$$F(x;\lambda) = 1 - e^{-\lambda x}, \quad x \geq 0$$

This function provides the probability that the time until the next event is less than or equal to x.

The simulation considers the particular queue model M/M/c where c =1 (M/M/1) because we consider that there are no alternatives for nodes. This can be adapted when the real situation indicates alternatives or not in graph nodes.

The code used for the simulation is presented below:

```
for _ in range(trials):
    propagation_results = []
    queue_lengths = {node: 0 for node in path}  # Track accumulated queue

    for i in range(len(path)-1):
        node, next_node = path[i], path[i + 1]

        # Node service rate
        service_rate = G.nodes[node].get('capacity', float('inf'))  # Default to 
no limit

        # Simulate arrival and service processes
        arrivals = poisson(arrival_lambda).rvs()
        service_times = expon(scale=mean_service_time).rvs(size=arrivals) if ar-
rivals > 0 else []

        # Process queue
        total_demand = arrivals + queue_lengths[node]
        if total_demand <= service_rate:
            queue_lengths[node] = 0
            status = "Processed"
        else:
            queue_lengths[node] = total_demand - service_rate
            status = "Overflow"

        # Edge latency handling
        edge_latency = G.edges[node, next_node].get('length', 1)  # Default la-
tency of 1 if missing

        propagation_results.append({
            'node': node,
            'arrivals': arrivals,
            'queue_length': queue_lengths[node],
            'status': status,
            'edge_latency': edge_latency
        })

    results.append(propagation_results)
```

3.4.5.4.5. Outputs

In our implementation, we have used Python scipy to simulate Poisson and exponential distributions.

Next, we print for each trial what is happening in each node:

Number of requests (arrivals), number of requests placed in queue, latency (wait time) and the status (process or overflow).

An example of a textual representation of values for each trial is:

```
Trial 200:
  Node Intersection9: Arrivals=44, Queue Length=0, Status=Processed, Edge La-
tency=497
  Node Intersection67: Arrivals=47, Queue Length=0, Status=Processed, Edge La-
tency=259
  Node Intersection68: Arrivals=58, Queue Length=9, Status=Overflow, Edge La-
tency=409
  Node Intersection70: Arrivals=46, Queue Length=0, Status=Processed, Edge La-
tency=213
```

3.4.5.4.6. User Interface

Besides this, we have plotted in a graph, for each trial, the number of requests in each node of the selected path. This is represented in the next figure (Figure 10 Queue length variation during trials).

On the X axis the number of trials is presented. On the Y axixs the queue length is represented.

In the figure: (Figure 10 Queue length variation during trials) we can see that there are 5 nodes on the path. In node Intersection 7 (orange) and Intersection 8 (green) is the biggest number of requests in the queue (more than 20). Considering the limit of 20 established at the beginning, this means that there are overflows.

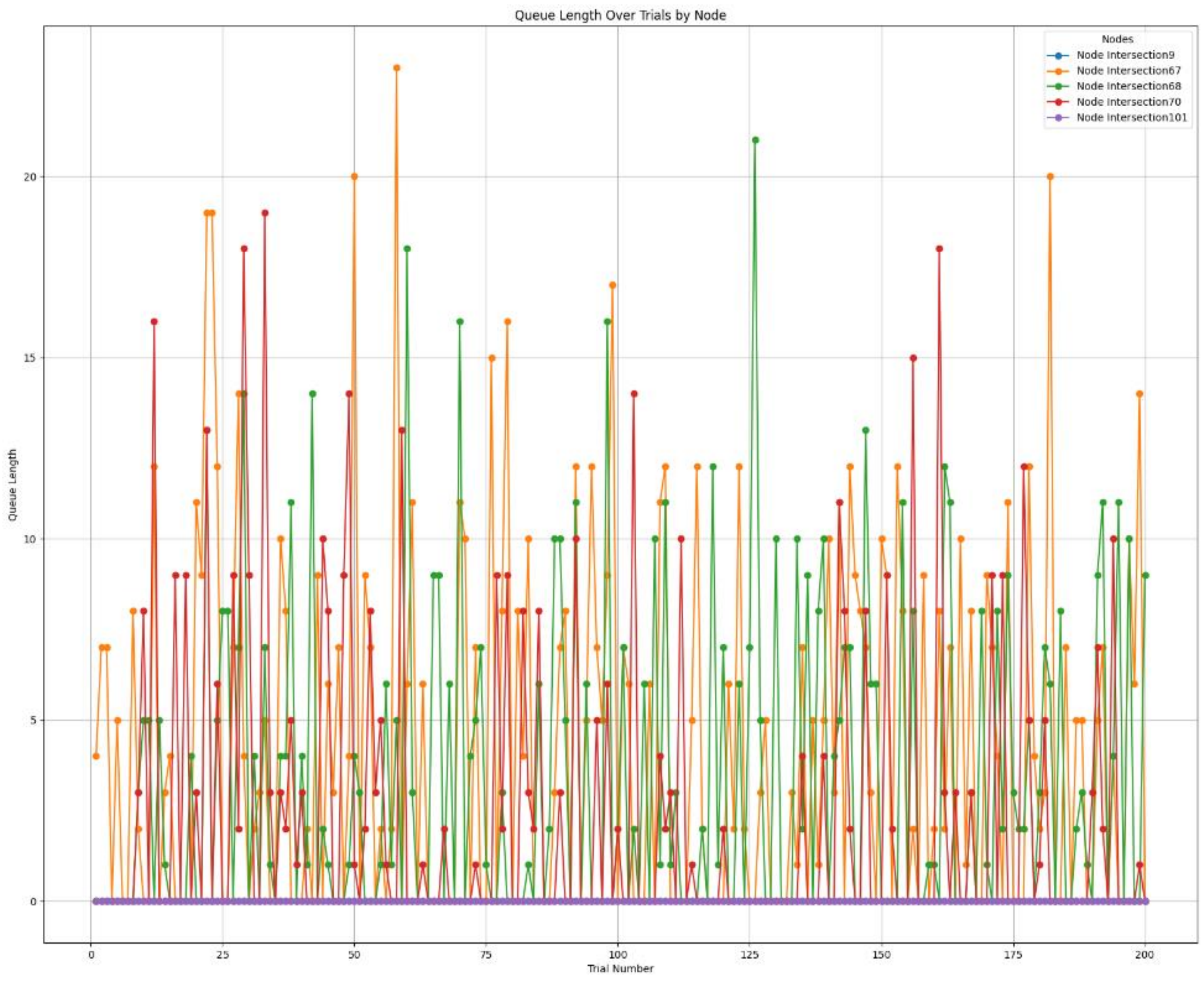


Figure 10 Queue length variation during trials

The graph allows users to identify intersections with the longest queues, enabling them to make informed decisions.
In addition to the graphical representation of paths on a map, it is also possible to call REST APIs that perform the same computations and return the paths as a list of nodes.

3.4.5.4.7. The REST APIs developed are:

```
@app.post("/simulate-queues")
async def api_simulate_queues(request: QueueSimulationRequest):
```

3.4.5.5 *Auto-Adaptive Traffic Control*

The final step involves running an auto-adaptive queue simulation. This process first initiates the simulation and then adjusts the node capacities to reduce queue lengths.

We have developed two methods, both utilizing Reinforcement Learning (RL), with the key difference being the reinforcement algorithm used.

3.4.5.5.1. Inputs

The inputs considered here are the same as the ones presented previously for the simulations plus the parameters necessary by the RL algorithms. They are:

1. Graph representation data obtained in the previous step. This should be in two csv files. One describing the nodes and one describing the edges. Example of such data:

Nodes file, a csv file with the content similar to:

```
osmid,y,x
26733684,37.9169119,23.7021355
26734032,37.9163994,23.7053716
26734041,37.9168317,23.70687
```

Edges file, a csv file with the content similar to:

```
u,v, name, length
26733684,7326480515,Unnamed Road,55.138041356770685
26733684,9018040029,Unnamed Road,91.7310126681424
26734032,4015873338,Ποσειδώνος,192.84253930809896
```

2. Start and end node.
   Example: `26733684, 26734032`
3. Simulation parameters (They define the type of algorithm used as described in the next paragraphs):

```
a. NO_TRIALS = 200            # Number of simulations run
b. ARRIVAL_LAMBDA = 50        # Requests per time unit
c. MEAN_SERVICE_TIME = 20      # average number of requests served in a
   time unit
d. QUEUE_CAPACITY = 20        # Maximum capacity of the queue
```

4. RL parameters

```
# Q-learning parameters
ALPHA = 0.1   # Learning rate
GAMMA = 0.9   # Discount factor
EPSILON = 0.1   # Exploration rate
NUM_EPISODES = 500   # Training episodes
```

These are general inputs from previous steps also. For this particular step, the output of the previous queue simulation is considered input. Also, the RL parameters.

3.4.5.5.2. Models and algorithms

In this step, we are comparing the results of the simulations described in the previous step, and the usage of RL to improve the graph. Improvement of the graph means changing the capacity for processing in each node.

We are referring here just to the RL part.

The algorithm used is the RL algorithm called Q-Learning.

The Q-learning algorithm updates the Q-value using the formula:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q\,(s',a') - Q(s,a) \right]$$

Where

- α = **learning rate** (determines how much new information overrides the old value).
- $r + \gamma \max_{a'} Q\,(s',a')$ = **target Q-value**.
- $Q(s,a)$ = **current estimate**.
- The update rule **adjusts Q(s, a) toward a better estimate**.

The step-by-step process is:

1. Initialize the Q-table with zeros (or random values).
2. Choose an action aaa using an exploration-exploitation strategy (e.g., ε-greedy: take random action with probability ϵ, otherwise take the best-known action).
3. Take action, observe the reward r and the new state s'.
4. Update the Q-value using the formula above.
5. Repeat until convergence or a stopping condition (e.g., reaching an optimal policy).

For the implementation, we are using the following elements:

- **State**: Queue length at a node.
- **Actions**: Increase, maintain, or decrease capacity.
- **Reward**: Penalize long queues; reward efficient capacity usage.
- **Q-table Update**: Adjusts the policy dynamically over episodes.

The implementation in Python is:

```python
def get_state(node, queue_length):
    """Encodes the state as (node, queue_length)."""
    return (node, queue_length)


def choose_action(state):
    """Selects an action using an epsilon-greedy policy."""
    if state not in Q_table:
        Q_table[state] = [0, 0, 0]  # Actions: [Decrease, Maintain, Increase]

    if random.uniform(0, 1) < EPSILON:
        return random.choice([0, 1, 2])  # Explore (random action)
    else:
        return np.argmax(Q_table[state])  # Exploit (best action)


def update_Q_table(state, action, reward, next_state):
    """Updates the Q-table using the Bellman equation."""
    if next_state not in Q_table:
        Q_table[next_state] = [0, 0, 0]

    best_next_action = np.max(Q_table[next_state])
    Q_table[state][action] += ALPHA * (reward + GAMMA * best_next_action - Q_ta-
ble[state][action])



all_queue_lengths = {node: [0] * trials for node in path}
capacities = {node: 1 for node in path}  # Initial capacity

for episode in range(NUM_EPISODES):
    node_queues = {node: 0 for node in path}

    for trial in range(trials):
        for i in range(len(path) - 1):
            node, next_node = path[i], path[i + 1]
            service_rate = capacities[node]

            # Simulate arrivals & service
            arrivals = poisson(arrival_lambda).rvs()
            queue_length = max(0, arrivals + node_queues[node] - service_rate)
            node_queues[node] = queue_length

            # Store queue length
            all_queue_lengths[node][trial] = queue_length

            # RL State and Action Selection
            state = get_state(node, queue_length)
            action = choose_action(state)

            # Perform Action (Modify Capacity)
            if action == 0 and capacities[node] > 1:  # Decrease capacity
                capacities[node] -= 1
            elif action == 2:  # Increase capacity
                capacities[node] += 1

            # Reward: Encourage queue reduction, penalize long queues
            reward = -queue_length if queue_length > 5 else 10 - queue_length

            # Next state
            next_state = get_state(node, queue_length)
            update_Q_table(state, action, reward, next_state)

    if episode % 100 == 0:
        print(f"Episode {episode}: Capacities - {capacities}")
```

3.4.5.5.3. Outputs

The outputs are the new capacities proposed at each running episode so that the queue length decreases.

For example:

```
Episode 10: Capacities - {'Intersection1': 1, 'Intersection41': 1, 'Intersec-
tion46': 1, 'Intersection101': 1}
Episode 20: Capacities - {'Intersection1': 1, 'Intersection41': 1, 'Intersec-
tion46': 3, 'Intersection101': 1}
Episode 30: Capacities - {'Intersection1': 1, 'Intersection41': 1, 'Intersec-
tion46': 1, 'Intersection101': 1}
Episode 40: Capacities - {'Intersection1': 4, 'Intersection41': 1, 'Intersec-
tion46': 1, 'Intersection101': 1}
```

Here we see the capacities adjusted in each node after running the algorithm.

3.4.5.5.4. User interface

When executing the RL method, the results are shown in next figure (Figure 11 Queue length after RL method 1)

On the X axis the number of trials is presented. On the Y axixs the queue length is represented.

On the right side, the queue length before adaptation is displayed, where all nodes in the paths have queue lengths ranging between 30 and 70.

After applying the RL method, the queue length is reduced to a range of 0 to 30, demonstrating a significant improvement.

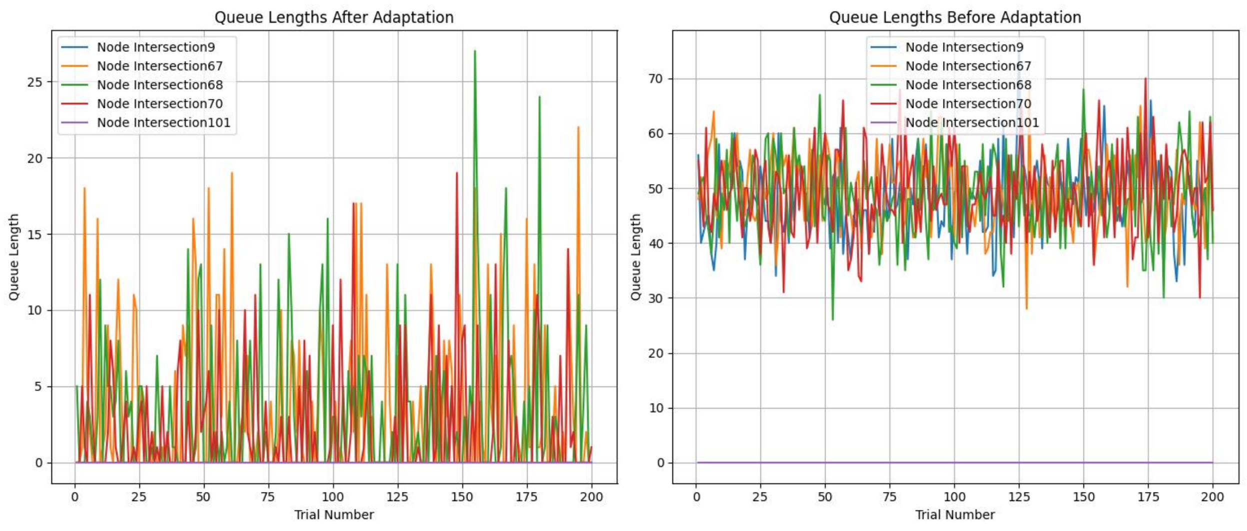


Figure 11 Queue length after RL method 1

Finally, the locations with the longest queues are displayed on the map, allowing the user to make informed decisions on traffic management and organization (Figure 12 Intersections with the heaviest traffic).

The length of the queue is also represented on the map. The bigger the node (blue circle) is, the bigger the queue length.

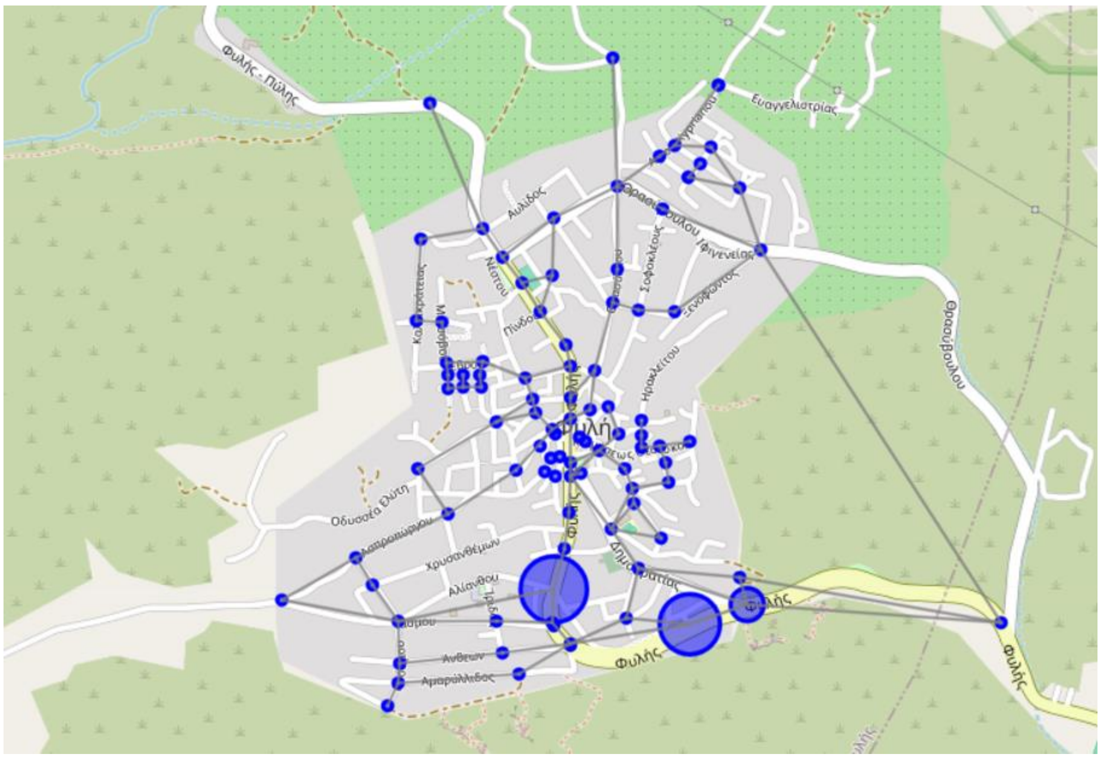


Figure 12 Intersections with the heaviest traffic

3.4.5.5.5. REST APIs

The REST APIs developed are:

```
@app.post("/simulate-queues")
async def api_simulate_queues(request: QueueSimulationRequest):
```

3.4.6 INTERPRETATION OF THE DATA, AND RESULTS

The system adapts in real-time based on seismic and environmental conditions, using reinforcement learning (RL) and rule-based adaptation to optimize response strategies. It uses different datasets and offers a decision support mechanism for the personnel involved in the management of such disasters. In summary, we can present the following data and presentations offered:

- Data on **natural gas stations, pipelines, railways, telecommunications, and power grids** is used to prioritize road closures and ensure emergency access to critical locations. This information is placed on nodes.
- The system dynamically reroutes traffic using **graph-based shortest and independent path algorithms** when infrastructure failures occur.
- Reinforcement learning optimizes traffic signals and lane allocations to **minimize congestion during mass evacuations**.
- AI adapts emergency response routes for agencies such as the **Hellenic Fire Service, Hellenic Police, and emergency medical teams**.
- AI algorithms adjust infrastructure risk models dynamically, ensuring **real-time recalibration** of evacuation and traffic control strategies.

To enhance real-time decision-making, the system provides:

- **Charts, graphs, and maps** provide real-time situational awareness for decision-makers.
- **REST APIs** expose auto-adaptive services for seamless integration with emergency management platforms.

3.5 SELF-ADAPTIVE HEATWAVE MODELS

Our approach presents an AI-enabled auto-adaptive traffic simulation that models human behaviour and vehicle movement in the situation of a heatwave, using OpenStreetMap[9] data. The system creates a graph representation of the road network, dynamically computes shortest paths, analyses congestion, and applies adaptive rules. The auto-adaptive AI framework dynamically optimizes evacuation routes, hospital resource allocation, and emergency response coordination, ensuring a resilient, data-driven approach to complex scenarios.

3.5.1 KEY FACTORS IN HEATWAVE SIMULATION

For the specific scenario of a heatwave in Vienna, the system focuses on managing **ambulance traffic, hospital occupancy, and healthcare resource allocation**.

Traffic Simulation for Ambulance Routes: The main intersection points are considered, and a graph is created to represent them and the roads between them

Hospital Triage and Capacity Management: A three-level graph representation models patient flow:

- First Level**:** Call origin (incident location).
- Second Level: Triage stations.
- Third Level: Hospitals.

Queue simulations assess how triage and hospital intake capacities affect response efficiency.

Auto-adaptive AI optimizes triage and hospital capacities in real-time to balance patient distribution and prevent overload.

Integration with Disaster Response Frameworks

The **heatwave models** align through shared AI-driven response mechanisms:

- Critical infrastructure interdependencies: The models assess how heat stress impacts power supply, hospitals, and transportation networks.
- Adaptive traffic control: AI dynamically adjusts road usage for ambulances and emergency responders based on hospital capacities and triage needs.
- Evacuation coordination: Uses historical and real-time hazard data to prioritize safe zones and emergency healthcare staging areas.

[9] https://www.openstreetmap.org/

3.5.2 USE CASES

The use cases specific to the earthquake scenario are presented in the next figure extracted from D3.6.

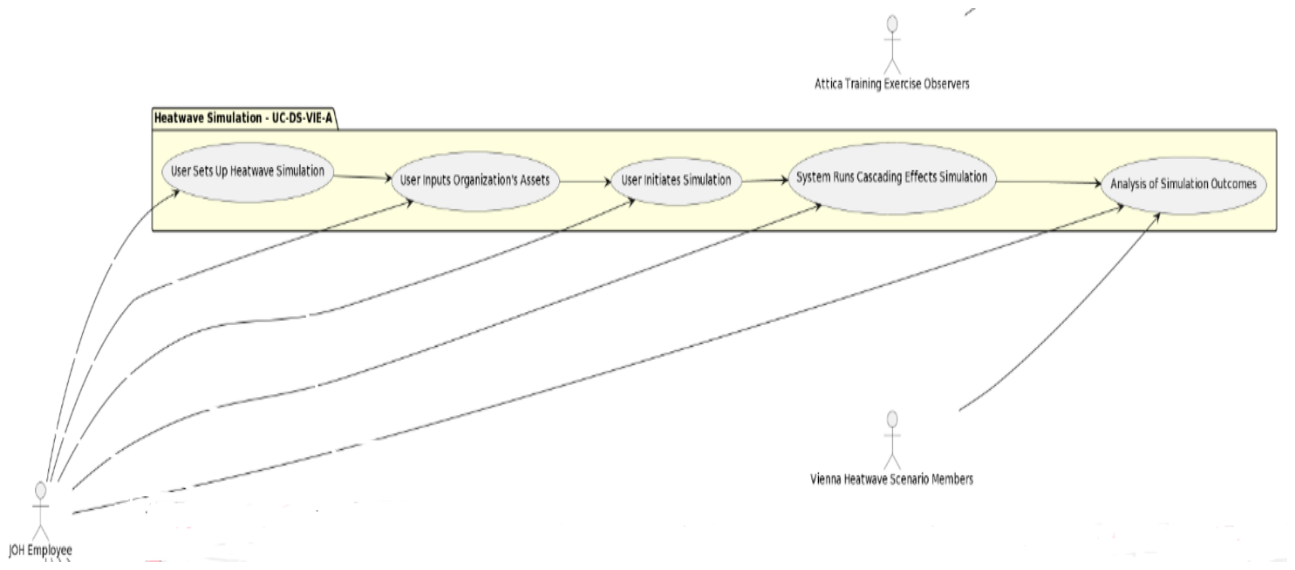

Attica Training Exercise Observers
Heatwave Simulation - UC-DS-VIE-A
User Sets Up Heatwave Simulation
User Inputs Organization's Assets
User Initiates Simulation
System Runs Cascading Effects Simulation
Analysis of Simulation Outcomes
Vienna Heatwave Scenario Members
JOH Employee

Figure 13 Heatwave simulation use cases (as in D3.6)

This diagram describes a heatwave preparedness simulation, allowing users to configure a realistic heatwave scenario, run the simulation, and analyse the cascading effects. Stakeholders gain valuable insights into the impacts of heatwaves on resources, infrastructure, and public health, enabling them to refine response strategies and enhance readiness for future incidents. This simulation provides a comprehensive tool for training, resource allocation, and decision-making in heatwave response scenarios.

UC-DS-VIE-A-1-User Sets Up Heatwave Simulation:

In the initial step, a user, specifically a JOH Employee, configures the heatwave simulation environment. This setup typically involves defining key parameters such as temperature thresholds, geographic scope, duration, and affected populations or regions.

The parameter selection is based on the statistical analysis of existing data, which is identified through data visualization rather than an automated process due to the nature of the data. The studied data is sourced from ZAMG.

Given that heatwaves in this specific use case lack substantial statistical data, a general-purpose statistical model may be selected to supplement the analysis and obtain additional insights.

UC-DS-VIE-A-2-User Inputs Organization’s Assets:

The user inputs details about the assets available to respond to the heatwave. This may include cooling centres, medical staff, transportation, water distribution systems, and any other resources that could be deployed in response to extreme heat conditions. These inputs help the system simulate a realistic response based on the organization’s capacity. This use case is not affected by statistical analysis.

UC-DS-VIE-A-3-User Initiates Simulation:

After setting up the scenario and inputting organizational assets, the user initiates the simulation, which triggers the system to start the simulated heatwave event. Statistical analysis of data is indirectly used when parameters for simulation are established.

UC-DS-VIE-A-4-System Runs Cascading Effects Simulation:

The system processes the heatwave scenario, taking into account the effects of prolonged high temperatures on infrastructure, public health, and resources. Cascading effects might include increased health emergencies, demand for water and power, strain on medical resources, and other secondary impacts due to the heatwave. This use case is not affected by statistical analysis.

UC-DS-VIE-A-5-Analysis of Simulation Outcomes:

The final step involves analysing the simulation outcomes. Stakeholders such as Vienna Heatwave Scenario Members receive insights from the simulation. These insights may include data on asset utilization, effectiveness of response strategies, impact on affected populations, and potential improvements for future real-world scenarios. This analysis supports better preparedness and decision-making for heatwave-related incidents. This use case is indirectly affected by statistical analysis, considering that input parameters when established may use the results of statistical analysis.

3.5.3 SCENARIO

1. KPIs: The main target of the application of the algorithms is the establishment of the parameters (nodes, edges with their capacity, so that:
    a. The paths between two points exists if previously it did not exist.
    b. The paths length is the smallest possible. The KPI is the report between the length of the shortest path and the medium lengths of the paths.
    c. The queue length after applying the self adaption is smaller than the initial one. The KPI is the report between the maximum queue length after applying the algorithm pe maximum queue length before applying the algorithms.
2. **Initial Monitoring:** The model continuously receives temperature, humidity, and solar radiation data from weather stations and satellites, tracking local variations across urban and rural areas.
3. **Detecting Need for Adaptation:** The model detects a sudden decrease in humidity across a region, which could exacerbate heat conditions. Anomaly detection flags this change, prompting adaptation to adjust humidity parameters for more realistic heat index predictions. This is reflected in the graph representing the routes studied.
4. **Decision and Adaptation:** Based on previous adaptations to low humidity conditions, the model decides to increase local temperature projections and simulate higher heat stress levels for affected regions, changing the topology of routes.
5. **Predictive Proactive Adaptation:** Forecasts predict high-pressure conditions that could sustain extreme temperatures. The model proactively adjusts traffic to reflect the current situation.
6. **Feedback and Learning:** After the heatwave, the model compares its predictions to observed data, reinforcing successful adaptations and adjusting sensitivity to future changes.

3.5.4 AI METHODS FOR HEATWAVE MODEL ADAPTATION

The heatwave simulation is grounded in computational graph theory, queueing models, and reinforcement learning to analyse and adapt traffic movement in heatwave events. The approach is structured as follows:

Adaptive traffic control uses **rule-based** and **RL-based** queue length optimization.

Graph representation of critical routes or hospital access

- Data is collected from OpenStreetMap, where intersections are modelled as nodes and roads as edges in a graph representation.

- Graph-based traffic modelling considers **key intersections and critical hospital access roads**.

Path Computation and Adaptation

- The shortest paths between key locations are computed to simulate human movement and evacuation routes (Berge, 1982).
- Independent shortest paths are identified to prevent reliance on common intersections, reducing the risk of single points of failure.
- The system dynamically recomputes paths when intersections or roads become unavailable due to earthquake damage.

Traffic Simulation with Queue Theory

- Traffic congestion is modelled using M/M/c queueing systems, where intersections act as multi-server queues (Lapan, 2020).
- Queue lengths at intersections are computed to evaluate congestion levels and bottlenecks.
- Statistical distributions (Poisson, Weibull, Pareto, Exponential) (Davison, 2008) simulate stochastic variations in traffic flow, capturing real-world uncertainties.

Auto-Adaptive Traffic Control

- Rule-Based Adaptation: Intersection capacities are adjusted dynamically using predefined heuristics to reduce queue lengths.
- Reinforcement Learning (RL (Q-Learning) (Lapan, 2020)): AI learns optimal traffic management strategies by continuously adjusting intersection and road capacities based on observed traffic patterns.
- Adaptive controls ensure that congested roads and intersections dynamically evolve based on real-time conditions.

Data Storage and Analysis

- Results, including nodes (intersection capacities) and edges (road lengths), are stored in CSV format for future simulations and analysis.
- If no feasible evacuation route is found, the system notifies users, enabling proactive disaster response planning.

Output Representation and API Integration

- **Visualization:** Results are displayed through **charts, graphs, and maps**, providing intuitive insights into congestion, queue lengths, and alternative routes.
- **REST API Exposure:** All functionalities, including path computation, queue length analysis, and adaptive traffic control, are exposed as **RESTful APIs**, enabling seamless integration with external disaster management and urban planning systems.

This computational framework ensures a real-time, adaptive, and data-driven approach to heatwave-related traffic management, improving urban mobility under seismic stress conditions.

3.5.4.1 *User interface and services offered*

The simulation output is either displayed on a local canvas or saved as a PNG file when it involves graphs and charts, allowing it to be accessed in other applications. Map-based information is presented as HTML pages, which are also stored locally for future reference.

Additionally, the simulation can be executed via a REST API, generating one of the following outputs:

- A PNG file visualizing the fire representation,
- An HTML file displaying information on a map, or
- A JSON file containing the final computed parameters after applying the adaptive algorithm.

Both the user interface and the REST API will follow the main steps required to run the simulation.

3.5.4.2 *Graph representation of critical routes or hospital access*

This is the first step in our simulation model. We are building here the graph used to model the streets and the intersections. (Berge, 1982)

3.5.4.2.1. Inputs

The data used to define the route or hospital access is manually entered in csv files representing nodes and edges.
The data entered in csv files can be obtained after OpenStreetMap is called for all the roads and intersections. It is a subset of the OpenStreetMap data, and it refers only to the point of interest in our simulation.

An example of node definition is given below:

```
Node,Type,Latitude,Longitude,Capacity,No_Inputs,No_Outputs
Intersection1,intersection,38.107983,23.66651,51,True,False
Intersection6,intersection,38.104927,23.66902,88,False,False
Intersection9,intersection,38.102436,23.66903,85,False,False
```

An example of an edge definition is given below:

```
Start_Node,End_Node,Length,Travel_Time
Intersection1,Intersection6,488,14
Intersection6,Intersection9,488,14
Intersection9,Intersection16,526,16
Intersection16,Intersection101,736,14
```

The longitude and latitude coordinates are used to enable the visualization of points on a map.

3.5.4.2.2. Outputs

The output of this step is the representation of the graph using networkx[10] graph format.

3.5.4.2.3. User interface

Based on the data provided in the input files, a graph is generated and presented to the end user. An important aspect to note is the dimension of the nodes, which reflects their capacity. If the nodes represent intersections, the capacity indicates how many cars can pass through. If they represent triage points, the capacity corresponds to the triage capacity.

The final interpretation and application of this information are left to the user. (Figure 14 Graph representation of a critical route).

In the next figure the radius of the nodes is proportional with their capacity.

[10] https://networkx.org/

Red nodes are start and end point.

Blue nodes are important intersection points which should be monitored.

Green nodes are normal intersections.

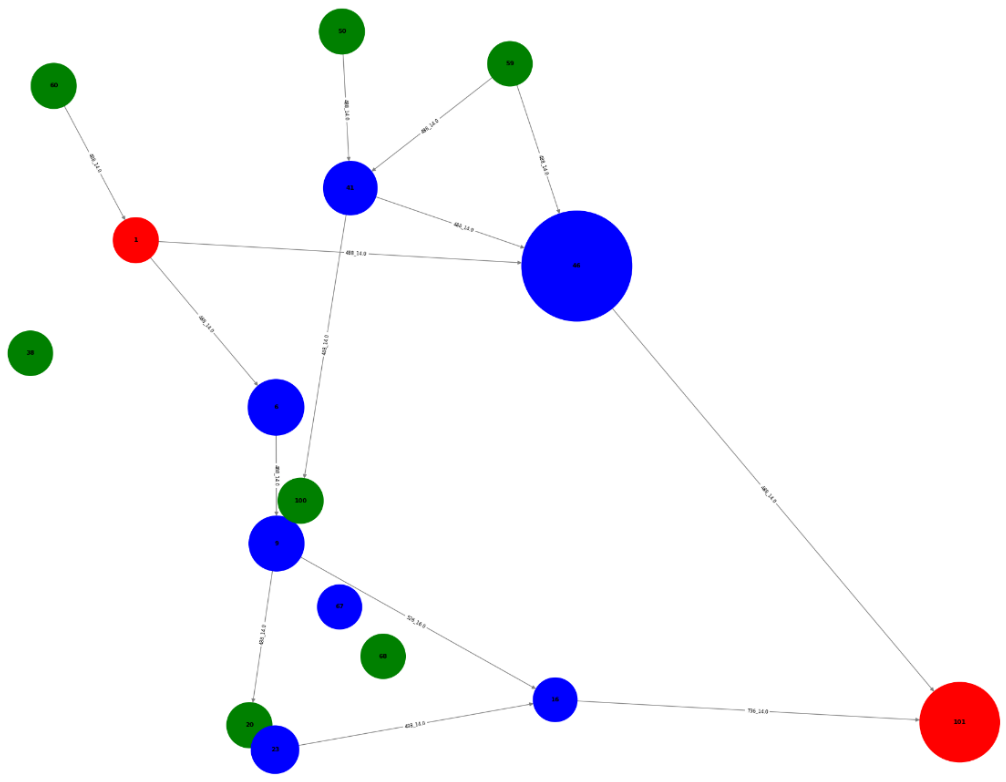


Figure 14 Graph representation of a critical route

3.5.4.2.4. REST APIs

The REST APIs developed are:

```
@app.post("/read-graph")
async def api_read_graph(request: GraphRequest):
```

3.5.4.3 *Optimum paths selection*

This is the second step and allows us to compute the paths between two points of interest.
After entering data for a specific area, the user selects two points: a Start Point and an End Point. The system then calculates the possible paths between these points.

3.5.4.3.1. Inputs

The inputs are:

The graph representation in networkx format, obtained from the previous step

Start point and end point (the name of the graph nodes), for example, `Intersection1`, `Intersection101`

3.5.4.3.2. Processing steps and methods:

The computation process includes the following steps:

1. *Cycle Removal* – Any cycles in the graph are eliminated to ensure valid paths.
2. *Shortest Path Calculation* – The system identifies the shortest paths between the selected points.
3. *Independent Path Computation* – Multiple independent paths are determined to provide alternative routing options.

1. *Cycle removal.*
   Like all other graph algorithms, the Python networkx library is used. Particularly for the cycle removal, the call of networkx[11] procedures is done in the following code:

```
while True:
    try:
        # Attempt a topological sort
        cycle=nx.find_cycle(graph, orientation='original')
        # If a cycle exists, remove one edge from the cycle

        graph.remove_edge(*cycle[-1][:2])

    except nx.NetworkXNoCycle:
        # No cycles remain
        break
```

`find_cycle` method is implemented in networkx using a cycle search via depth-first traversal.

2. *Find the paths between the two nodes in increasing total length*

   After removing the cycles, the paths between the start and end node are computed. Input is the graph resulting after cycle removal.

   The Python code, calling networkx graph procedures is:

```
all_paths = list(nx.all_simple_paths(graph, source=source, target=target))
# A better variant is to use a method based on Djikstra algorithm
# all_paths = list(nx.all_shortest_paths(graph, source=source, target=tar-
get))

path_lengths = [(path, sum(graph[u][v]['length'] for u, v in zip(path[:-1],
path[1:]))) for path in all_paths]
sorted_paths = sorted(path_lengths, key=lambda x: x[1])
```

   The algorithm used by the all_simple_paths method is a modified depth-first search to generate the paths[12].

[11] https://networkx.org/
[12] https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.simple_paths.all_simple_paths.html

The algorithm used by the all_shortest_paths method is a modified Dijkstra's algorithm to search the paths[13].

When the graph is simple (hundred of nodes) `all_simple_paths` method is used as it is fast and gives all the paths.

When the graph becomes more complex then it is recommended first to test if there exist paths between the start and end node, then run several times `all_shortest_path` to get the required paths.

3. *Find independent paths between the two nodes.*

Having as input all the paths detected between the start and end node, in the next step only the independent paths are computed.

The code used is:

```
# Filter independent paths
independent_paths = []
used_nodes = set()
for path, length in sorted_paths:
if not any(node in used_nodes for node in path[1:-1]):
        independent_paths.append(path)
        used_nodes.update(path[1:-1])  # Avoid reusing intermediate nodes

return independent_paths
```

If no paths are available, the system displays notifications to inform the user.

3.5.4.3.3. Outputs

The computed results are:

- Displayed on a map for visual interpretation.
- Saved as an HTML file for future reference.

3.5.4.3.4. User interface

The paths from one selected point to another are visually represented as shown in the following figure. (Figure 15 All paths and independent paths between two nodes)

In the next figure the radius of the nodes is proportional with their capacity.

Red nodes are start and end point.

Blue nodes are important intersection points which should be monitored.

Green nodes are normal intersections.

Magenta edges are paths respective independent paths between start and end point.

13 https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.shortest_paths.generic.all_shortest_paths.html#networkx.algorithms.shortest_paths.generic.all_shortest_paths

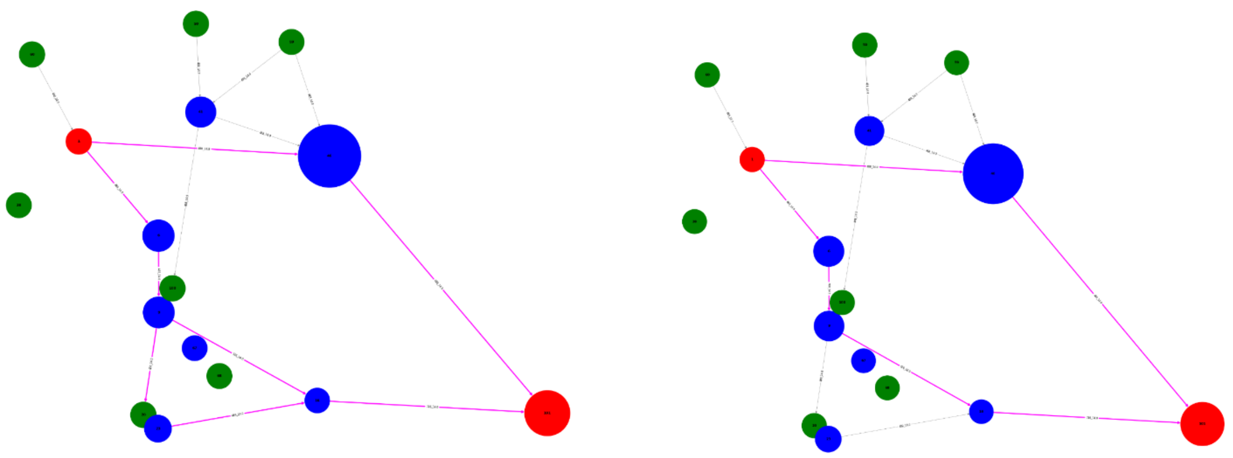


Figure 15 All paths and independent paths between two nodes

3.5.4.3.5. REST APIs

The REST APIs used are:

```
@app.post("/compute-shortest-path")
async def api_compute_shortest_path(request: PathRequest):


@app.post("/compute-independent-path")
async def api_compute_independent_path(request: PathRequest):
```

3.5.4.4 *Queues simulation*

3.5.4.4.1. Inputs

1. The paths obtained in the previous step are in networkx format. For example:
   ```
   a. Path: ['Intersection1', 'Intersection50', 'Intersection59', 'Inter-
      section101']
   ```
2. Simulation parameters (They define the type of algorithm used as described in the next paragraphs):
   ```
   a. ARRIVAL_RATE = 10  # average number of calls per minute (λ)
   b. SERVICE_RATE = 2  # average number of calls served per minute per
      server (μ)
   c. NUM_SERVERS = 5  # number of call agents (servers)
   d. SIM_TIME = 1000  # simulation time in minutes
   ```

3.5.4.4.2. Models and algorithms

Now, with the computed paths and independent paths, we can simulate the requested access using queue theory. Specifically, we employ an M/M/c queue model.

The notation M/M/c follows Kendall's notation [(Lee, 1966)], where:

- M (Markovian arrivals) = The inter-arrival times follow an exponential (Poisson) process with rate λ (ARRIVAL_LAMBDA input parameter)

- M (Markovian service times) = The service times are exponentially distributed with rate μ per server (MEAN_SERVICE_TIME input parameter).
- c (Number of servers) = There are ccc servers in parallel, all serving customers from a single queue (Tested for 1 and 2).
- K (Queue capacity) = Maximum capacity of a queue (QUEUE_CAPACITY input parameter)

The requests simulated represent:

- "Ambulances" navigating through different intersections, or
- Patients requesting assistance at triage points.

Each intersection has a defined capacity:

- For intersections, this represents the number of ambulances that can pass within a given time interval.
- For triage points, this corresponds to the number of patients that can be processed.

3.5.4.4.3. Performance metrics

For the simulation considered, we can compute the following performance metrics:

Expected Number of Requests in System (Little's Law)

Using **Little's Law** (L=λW), the average number of requests in the entire system (queue + service) is:

$$L = L_q + \frac{\lambda}{\mu}$$

Where:

- L_q is the average number of requests in the queue.

Expected Number of Requests in Queue

The expected number of requests waiting in the queue **is:**

$$L_q = \frac{P_w \cdot \rho \cdot c}{1 - \rho_c}$$

The elements in the formula are:

- L_q (Expected Number of requests in Queue)

  This represents the average number of requests waiting in the queue before service (excluding those currently being served).

  It is a key performance metric to measure congestion in the system.

- P_w (Probability That an Arriving Request Has to Wait)

  This is the probability that all c servers are busy, meaning an arriving request must wait in the queue before being served.

  It is computed using the Erlang C [(Lee, 1966)] formula:

$$P_w = \frac{(\lambda/\mu)^c}{c!} \cdot \frac{c\mu}{c\mu - \lambda} P_0$$

Where P_0 is the probability that there are zero requests in the system and μ is the service rate.

- ρ (Traffic Intensity per Server)

  It is the ratio of arrival rate to service rate per server:

$$\rho = \frac{\lambda}{c\mu}$$

This represents the fraction of time that each server is busy, also known as server utilization.

- c (Number of Servers)

This is the total number of servers available in the system.

It affects the queue length because having more servers reduces waiting times.

- ρ_c (Overall System Utilization)

The system utilization is given by:

$$\rho_c = \frac{\lambda}{c\mu}$$

This represents the fraction of the time that the system (all servers combined) is busy.

When ρ_c approaches 1 (i.e., the system is highly utilized), waiting times increase significantly.

- $1-\rho_c$ (Idle Time Factor)

This represents the probability that at least one server is idle at any given moment.

It acts as a denominator to normalize the equation and prevent infinite values when the system is heavily loaded.

Average Waiting Time in Queue

The average waiting time in the queue (before being served) is:

$$W_q = \frac{L_q}{\lambda}$$

The components of the formula are explained in the previous parameter description.

Average Time Spent in the System

The total time a customer spends in the system (waiting + service time) is:

$$W = W_q + \frac{1}{\mu}$$

For example, on the first try, we have:

Inputs

Simulation time: 1000 minutes

Number of servers: 2

Arrival rate: 10 calls/minute

Service rate: 2 calls/minute per server

Outputs

Average waiting time in queue: **311.5594** minutes

Maximum waiting time in queue**: 614.2010** minutes

Average time in system (waiting + service): **311.9135** minutes

Approximate server utilization**: 0.9993**

In a situation when the increased from 2 to 5, we have:

Inputs

Simulation time: 1000 minutes

Number of servers: 5

Arrival rate: 10 calls/minute

Service rate: 2 calls/minute per server

Outputs

Average waiting time in queue: **6.5923** minutes

Maximum waiting time in queue: **20.4640** minutes

Average time in system (waiting + service): **7.0921** minutes

Approximate server utilization: 0.**9842**

3.5.4.4.4. Outputs

After running the simulation, we obtain:

- A cumulative description of queue lengths, For example:

```
  Node Intersection1: Arrivals=51, Queue Length=0, Status=Processed, Edge Du-
ration=431
  Node Intersection50: Arrivals=37, Queue Length=0, Status=Processed, Edge
Duration=431
  Node Intersection59: Arrivals=47, Queue Length=23, Status=Overflow, Edge
Duration=931
  Node Intersection101: Arrivals=44, Queue Length=0, Status=Processed, Edge
Duration=431
```

- A graphical representation of the results is shown in the following figure. (Figure 16 Queue length):

3.5.4.4.5. User interface

Besides this, we have plotted in a graph, for each trial, the number of requests in each node of the selected path. This is represented in the next figure (Figure 16 Queue length).

On the X axis the number of trials is represented.

On the Y axis the queue length is represented.

We can see in the figure the nodes in the paths, and how many requests are in the queue.

In our example, the node "Intersection59" is the one with many requests in the queue. The others do not have queues. To solve the problem, we have to treat (increase capacity, create another path, etc) for Intersection 59.

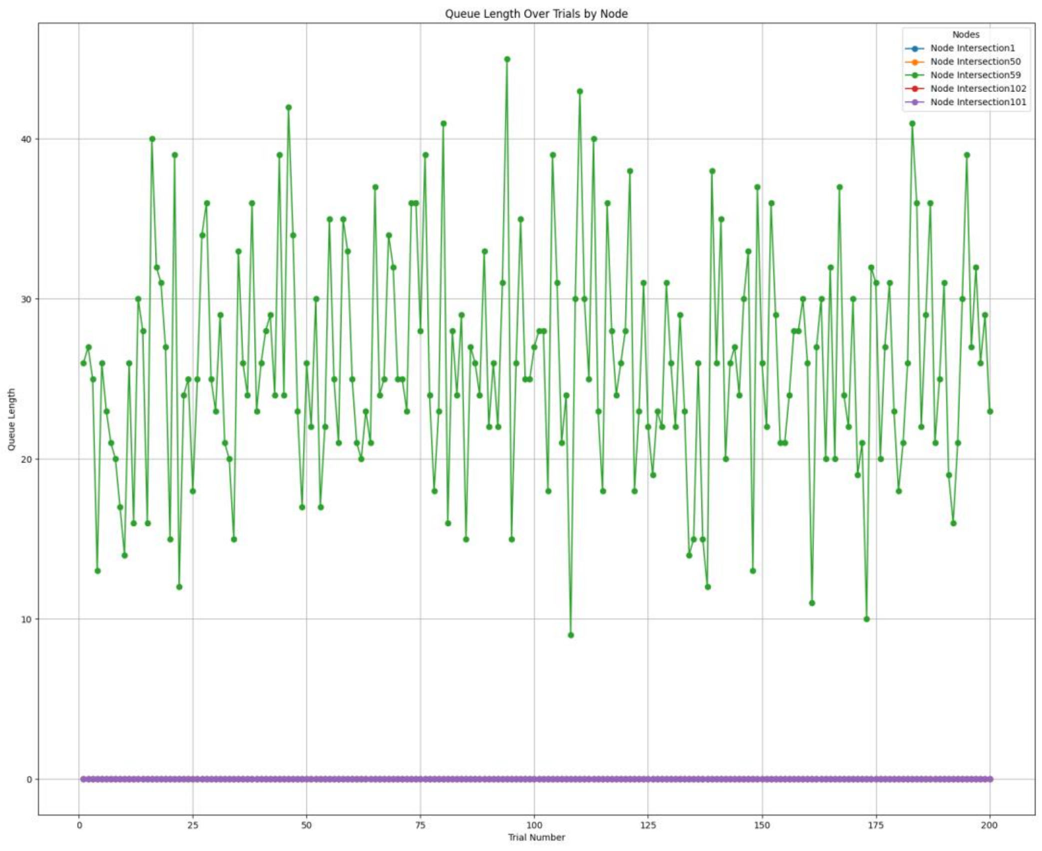


Figure 16 Queue length

3.5.4.4.6. REST API

The REST APIs used are:

```
@app.post("/simulate-queues")
async def api_simulate_queues(request: QueueSimulationRequest)
```

3.5.4.5 *Auto adaptive queue simulation*

The final step involves running an auto-adaptive queue simulation. This process first initiates the simulation and then dynamically adjusts the node capacities to reduce queue lengths.

3.5.4.5.1. Inputs

1. The paths obtained in the previous step are in networkx format. For example:

```
a. Path: ['Intersection1', 'Intersection50', 'Intersection59', 'Inter-
   section101']
```

2. Simulation parameters (They define the type of algorithm used as described in the next paragraphs):

```
a. ARRIVAL_RATE = 10   # average number of calls per minute (λ)
b. SERVICE_RATE = 2  # average number of calls served per minute per
   server (μ)
c. NUM_SERVERS = 5  # number of call agents (servers)
d. SIM_TIME = 1000  # simulation time in minutes
```

5. RL parameters

```
# Q-learning parameters
ALPHA = 0.1   # Learning rate
GAMMA = 0.9   # Discount factor
EPSILON = 0.1   # Exploration rate
NUM_EPISODES = 500   # Training episodes
```

3.5.4.5.2. Models and algorithms

In this step, we are comparing the results of the simulations described in the previous step, and the usage of RL to improve the graph. Improvement of the graph means changing the capacity for processing in each node.

We are referring here just to the RL part.

The algorithm used is the RL algorithm called Q-Learning.

The Q-learning algorithm updates the Q-value using the formula:

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right]$$

Where

- α = **learning rate** (determines how much new information overrides the old value).
- $r + \gamma \max_{a'} Q(s',a')$ = **target Q-value**.
- $Q(s,a)$ = **current estimate**.
- The update rule **adjusts Q(s, a) toward a better estimate**.

The step-by-step process is:

6. **Initialize** the Q-table with zeros (or random values).
7. **Choose an action** aaa using an **exploration-exploitation strategy** (e.g., ε-greedy: take random action with probability ϵ, otherwise take the best-known action).
8. **Take action**, observe the **reward** r and the **new state** s'.
9. **Update** the Q-value using the formula above.
10. **Repeat** until convergence or a stopping condition (e.g., reaching an optimal policy).

For the implementation, we are using the following elements:

- **State**: Queue length at a node.
- **Actions**: Increase, maintain, or decrease capacity.
- **Reward**: Penalize long queues; reward efficient capacity usage.
- **Q-table Update**: Adjusts the policy dynamically over episodes.

The implementation in Python is:

```
def get_state(node, queue_length):
    """Encodes the state as (node, queue_length)."""
    return (node, queue_length)


def choose_action(state):
    """Selects an action using an epsilon-greedy policy."""
    if state not in Q_table:
        Q_table[state] = [0, 0, 0]  # Actions: [Decrease, Maintain, Increase]

    if random.uniform(0, 1) < EPSILON:
        return random.choice([0, 1, 2])  # Explore (random action)
    else:
        return np.argmax(Q_table[state])  # Exploit (best action)


def update_Q_table(state, action, reward, next_state):
    """Updates the Q-table using the Bellman equation."""
    if next_state not in Q_table:
        Q_table[next_state] = [0, 0, 0]

    best_next_action = np.max(Q_table[next_state])
    Q_table[state][action] += ALPHA * (reward + GAMMA * best_next_action - Q_ta-
ble[state][action])



all_queue_lengths = {node: [0] * trials for node in path}
capacities = {node: 1 for node in path}  # Initial capacity

for episode in range(NUM_EPISODES):
    node_queues = {node: 0 for node in path}

    for trial in range(trials):
        for i in range(len(path) - 1):
            node, next_node = path[i], path[i + 1]
            service_rate = capacities[node]

            # Simulate arrivals & service
            arrivals = poisson(arrival_lambda).rvs()
            queue_length = max(0, arrivals + node_queues[node] - service_rate)
            node_queues[node] = queue_length

            # Store queue length
            all_queue_lengths[node][trial] = queue_length

            # RL State and Action Selection
            state = get_state(node, queue_length)
            action = choose_action(state)

            # Perform Action (Modify Capacity)
            if action == 0 and capacities[node] > 1:  # Decrease capacity
                capacities[node] -= 1
            elif action == 2:  # Increase capacity
                capacities[node] += 1

            # Reward: Encourage queue reduction, penalize long queues
            reward = -queue_length if queue_length > 5 else 10 - queue_length

            # Next state
            next_state = get_state(node, queue_length)
            update_Q_table(state, action, reward, next_state)

    if episode % 100 == 0:
        print(f"Episode {episode}: Capacities - {capacities}")
```

3.5.4.5.3. Outputs

The outputs are the new capacities proposed at each running episode so that the queue length decreases.

For example:

```
Episode 60: Capacities - {'Intersection1': 2, 'Intersection41': 2, 'Intersec-
tion46': 7, 'Intersection101': 1}
Episode 70: Capacities - {'Intersection1': 1, 'Intersection41': 7, 'Intersec-
tion46': 1, 'Intersection101': 1}
Episode 80: Capacities - {'Intersection1': 3, 'Intersection41': 10, 'Intersec-
tion46': 8, 'Intersection101': 1}
```

3.5.4.5.4. User interface

When executing the simulations, the results are shown in Figure 17 Adapted queue length(I)

- On the right side, the queue length before adaptation is displayed, showing values between 30 and 65 for all nodes in the paths.
- After applying RL, the queue length reduces to 0, demonstrating a significant improvement in efficiency.

A smaller queue lengths indicate that the system is behaving better (smaller waiting time).

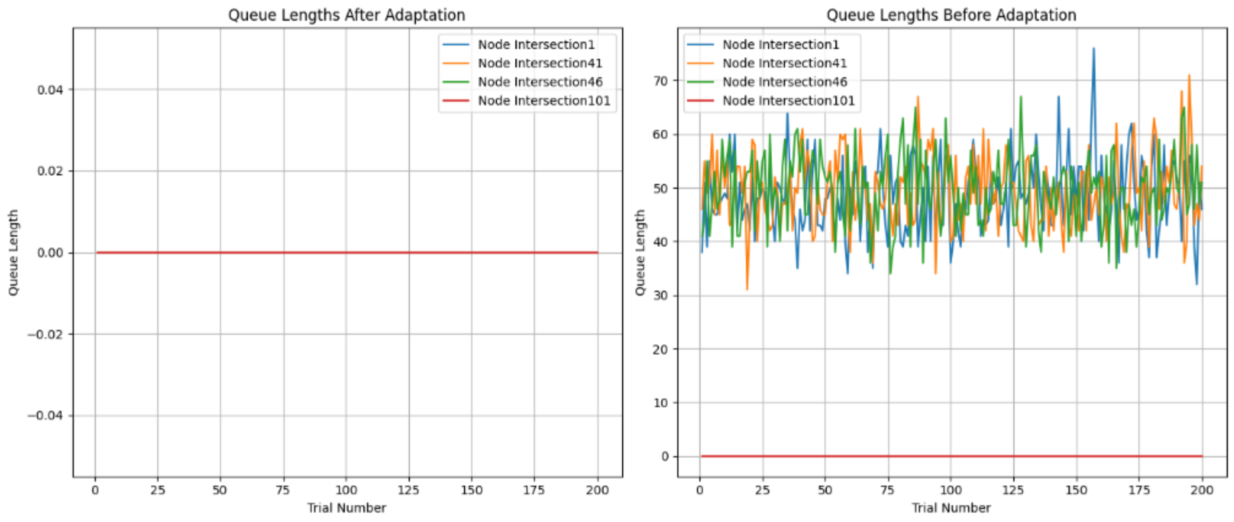


Figure 17 Adapted queue length(I)

3.5.4.5.5. REST API

The REST APIs used are:

```
@app.post("/simulate-queues-adaptive-a")
async def api_simulate_queues_adaptive(request: QueueSimulationRequest):


@app.post("/simulate-queues-adaptive-b")
async def api_simulate_queues_adaptive(request: QueueSimulationRequest):
```

3.5.5 INTERPRETATION OF THE DATA, AND RESULTS

The system adapts in real-time based data in traffic or triage points, using reinforcement learning (RL) and rule-based adaptation to optimize response strategies. It uses different datasets and offers a decision support mechanism for the personnel involved in the management of such disasters. In summary, we can present the following data and presentations offered:

- The system dynamically reroutes traffic using **graph-based shortest and independent path algorithms** when infrastructure failures occur.
- Reinforcement learning optimizes traffic signals and lane allocations to **minimize congestion during mass evacuations**.
- AI adapts emergency response routes for agencies such as the **Police, and emergency medical teams**.
- AI algorithms adjust infrastructure risk models dynamically, ensuring **real-time recalibration** of evacuation and traffic control strategies.

To enhance real-time decision-making, the system provides:

- **Charts, graphs, and maps** provide real-time situational awareness for decision-makers.
- **REST APIs** expose auto-adaptive services for seamless integration with emergency management platforms.

3.6 SELF-ADAPTIVE MAN-MADE DISASTER MODELS

Man-made disasters, such as toxic gas leaks, industrial accidents, or infrastructure failures, require adaptive and intelligent response mechanisms. Auto-adaptive AI simulations integrate techniques previously applied to wildfires, earthquakes, and heatwaves, ensuring effective disaster response, resource allocation, and risk mitigation. This approach enables real-time monitoring, decision-making, and adaptation through AI-driven methods, enhancing resilience and minimizing impact on human life and infrastructure.

3.6.1 KEY FACTORS IN MAN-MADE DISASTER SIMULATION

Toxic Gas Dispersion Modelling:

- Similar to fire spread simulations, the dispersion of toxic gases is modelled based on wind speed, temperature, and topographical conditions. The method used is based on the Lagrangian Particle Dispersion Model (LPDM) (Zannetti, 1990)
- Real-time sensor data from air quality monitoring stations could be integrated to update predictions dynamically.

Infrastructure Damage Assessment:

- Bridges, buildings, and transportation networks are analysed similarly to earthquake damage models.
- Graph-based simulations detect bottlenecks and failures, optimizing evacuation paths and rerouting emergency response vehicles.

Health and Medical Response Optimization:

- Ambulance dispatch and hospital capacity management follow the same principles as in the heatwave scenario.
- AI-driven queue models ensure rapid triage and optimal resource allocation at hospitals and emergency centres.

3.6.2 USE CASES

The use cases specific to the man-made disaster scenario are presented in the next figure extracted from D3.6.

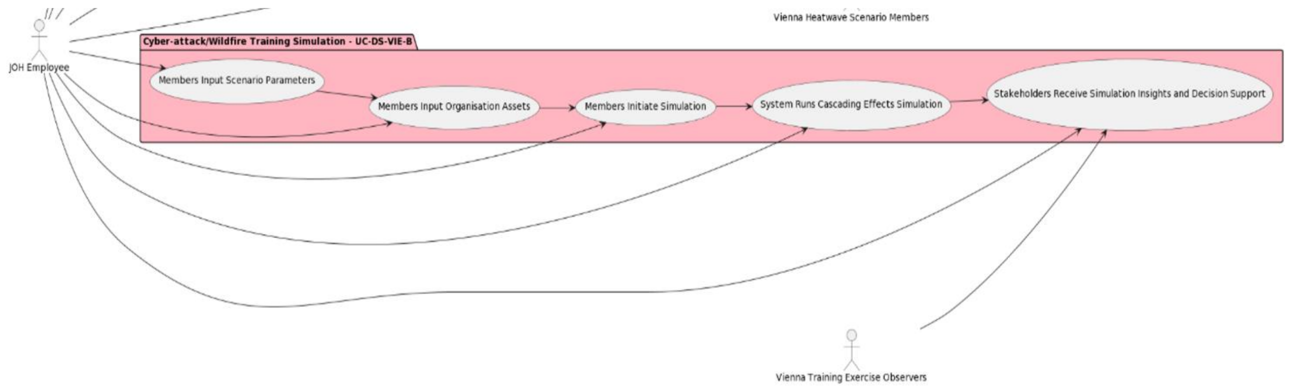

Vienna Heatwave Scenario Members
Cyber-attack/Wildfire Training Simulation - UC-DS-VIE-B
JOH Employee
Members Input Scenario Parameters
Members Input Organisation Assets
Members Initiate Simulation
System Runs Cascading Effects Simulation
Stakeholders Receive Simulation Insights and Decision Support
Vienna Training Exercise Observers

Figure 18 Man-made disaster simulation use cases (as in D3.6)

3.6.3 SCENARIO

1. **KPIs:** The simulation is based on stochastic model. It cannot be tested against real data. The dispersion is visually inspected, and its change checked when parameters are changed. It is a qualitative approach based on expert judgement.
2. **Initial Monitoring:** The model continuously receives data on chemical dispersion, wind speed, temperature, and population density, updating its predictions based on real-time sensor inputs.
3. **Detecting Need for Adaptation:** The model detects a wind shift that could spread the toxic plume toward residential areas. Anomaly detection flags this change, prompting an adaptation to adjust the plume's dispersion path and exposure predictions.
4. **Decision and Adaptation:** Based on hazard type and local geography, the model adapts its dispersion parameters to simulate faster spread along the new wind direction. It updates exposure estimates for areas now at risk.
5. **Proactive Prediction and Adaptation:** Forecasts predict rain, which could spread toxic runoff to nearby water sources. The model proactively adapts by simulating potential contamination paths in water systems and adjusting containment recommendations.
6. **Feedback and Learning:** After the incident, the model compares its predictions to actual containment data, reinforcing adaptations that accurately predict spread and risk, improving its response for future scenarios.

3.6.4 AI METHODS FOR MAN-MADE DISASTER MODEL ADAPTATION

Lagrangian Dispersion Models

- They describe fluid elements that follow the instantaneous flow. They simulate individual particles released into the atmosphere, tracking their movement with random fluctuations.

Graph Theory for Infrastructure & Traffic Management:

- Models affected road networks, hospitals, evacuation centres, and emergency response stations.
- Uses shortest-path algorithms and independent path calculations to optimize evacuation and rescue efforts.

Queue Theory for Medical and Traffic Simulations:

- M/M/c queue models analyse congestion in hospitals, triage centres, and ambulance dispatch locations.
- Adaptive queue adjustments (rule-based and RL-based) optimize emergency response efficiency.

Statistical Distributions for Risk and Response Prediction:

- Poisson distributions model the frequency of emergency calls and resource demands.
- Weibull and Pareto distributions predict structural failures in buildings and industrial sites.
- Exponential models estimate response times and hospital admissions.

Reinforcement Learning (RL) for Real-Time Adaptation:

- RL-based models adjust containment strategies, traffic rerouting, and medical dispatch coordination.

3.6.5 USER INTERFACE AND SERVICES OFFERED

For this scenario, there are two parts of the simulation. In the first part, the gas spread is simulated. This may imply the closure of some areas or routes. Consequently, in the second part we are treating the routes and paths between interest points, to simulate the situations when some areas are affected, and reaching them is not possible.

The simulation output is either displayed on a local canvas or saved as a PNG file when it involves graphs and charts, allowing it to be accessed in other applications. Map-based information is presented as HTML pages, which are also stored locally for future reference.

Additionally, the simulation can be executed via a REST API, generating one of the following outputs:

- A PNG file visualizing the fire representation,
- An HTML file displaying information on a map, or
- A JSON file containing the final computed parameters after applying the adaptive algorithm.

Both the user interface and the REST API will follow the main steps required to run the simulation.

3.6.5.1 *Gas spread simulation*

In the first part of the simulation, we are modelling the gas spread using the Lagrangian Particle Dispersion Model (LPDM)[(Zannetti, 1990)]

3.6.5.1.1. Inputs

1. The location where the simulation begins (OpenStreetMap coordinates)
2. Simulation parameters:

```
# Simulation parameters
num_particles = 5000     # Number of particles
source_position = (0, 0) # Source location (x, y)
wind_speed = 5           # m/s
wind_direction = 0       # Wind direction in degrees (0 = right, 90 = up)
diffusion_coeff = 0.5    # Turbulent diffusion coefficient
simulation_time = 100    # Time steps
dt = 1  # Time step (s)
```

3. Arrow keys are used to change the wind direction (in interactive mode)
4. + and – Keys used to increase/decrease the wind speed by 1 m/s (in interactive mode).

3.6.5.1.2. Processing steps and methods

We are simulating using the LPDM model.

The following assumptions are considered:

- Particles are massless points (mass is tracked independently).
- No particle-particle interactions.
- Turbulence is Gaussian and homogeneous (in basic form).
- Advection is linear across a small time step.

The Lagrangian Particle Dispersion Model is described as:

The position of a particle is updated over time as:

$$\vec{x}(t+\Delta t) = \vec{x}(t) + \vec{U}(\vec{x},t)\Delta t + \vec{u'}(\vec{x},t)\Delta t$$

Where:

- $\vec{x}(t)$: position vector at time (t)
- $\vec{U}(\vec{x},t)$: deterministic wind field (advection)
- $\overrightarrow{u'}(\vec{x},t)$: random component due to turbulence
- Δt: time step

The turbulent velocity $\overrightarrow{u'}$ is modelled as a stochastic (random) process using a random walk:

$$\overrightarrow{u'} = \sqrt{2K} \cdot \vec{\eta}$$

Where:

- K: turbulent diffusivity tensor (often simplified to scalar values in x, y, z)
- $\vec{\eta} \sim \mathcal{N}(0,1)$: Gaussian random vector

The wind field is:

U=(u,v,w)=(wind speed,0,0)

The concentration at any location is estimated using the particle count in grid cells:

$$C(x,y,z,t) = \frac{1}{\Delta V} \sum_{i=1}^{N} m_i \cdot \delta_{\Delta V}(\vec{x_i} - \vec{x})$$

Where:

- ΔV: volume of the grid cell
- m_i: mass per particle (constant or variable)
- $\delta_{\Delta V}$: binning function (1 if inside grid, 0 otherwise)

The implementation in Python is:

```python
def update_particles(particles, wind_speed, wind_direction, diffusion_coeff,
dt):
    """Move particles based on wind and diffusion."""
    theta = np.radians(wind_direction)
    wind_x = wind_speed * np.cos(theta) * dt
    wind_y = wind_speed * np.sin(theta) * dt

    # Random diffusion (Gaussian distribution)
    diff_x = np.random.normal(0, diffusion_coeff * np.sqrt(dt), len(particles))
    diff_y = np.random.normal(0, diffusion_coeff * np.sqrt(dt), len(particles))

    # Update particle positions
    particles[:, 0] += wind_x + diff_x
    particles[:, 1] += wind_y + diff_y
    return particles
```

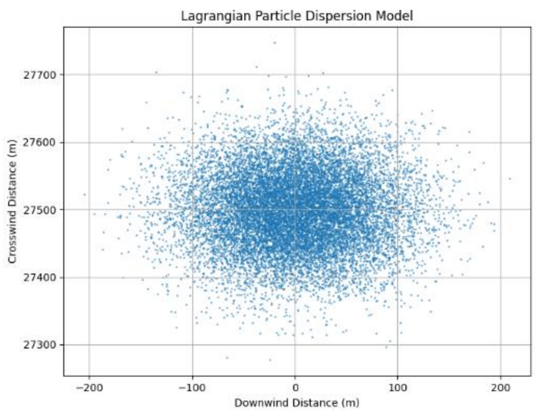
3.6.5.1.3. Outputs

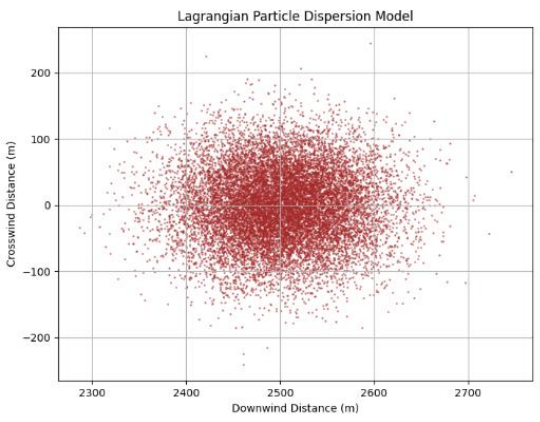
The simulation is displayed on a local canvas or saved as a PNG file, which can be opened in other applications.

For example, we have simulated a situation when there is no wind (wind_speed =0) and the situation with wind_speed = 5 m/s from the direction west (angle 0).

The first representation is displayed on the left of the next figure. The right part of the figure displays the situation when the wind moves the gas about 2000 m (see the numbers on the x horizontal scale in the charts)

The next outputs are working in non-interactive mode.





3.6.5.1.4. User interface

In interactive mode, the gas spread is displayed on the map. When pressing the arrow keys, the wind direction is changed. When pressing the +/- keys, the wind speed is changed and next another simulation is run.

An example of the output image is presented in the next figure (Figure 19 Gas spread simulation):

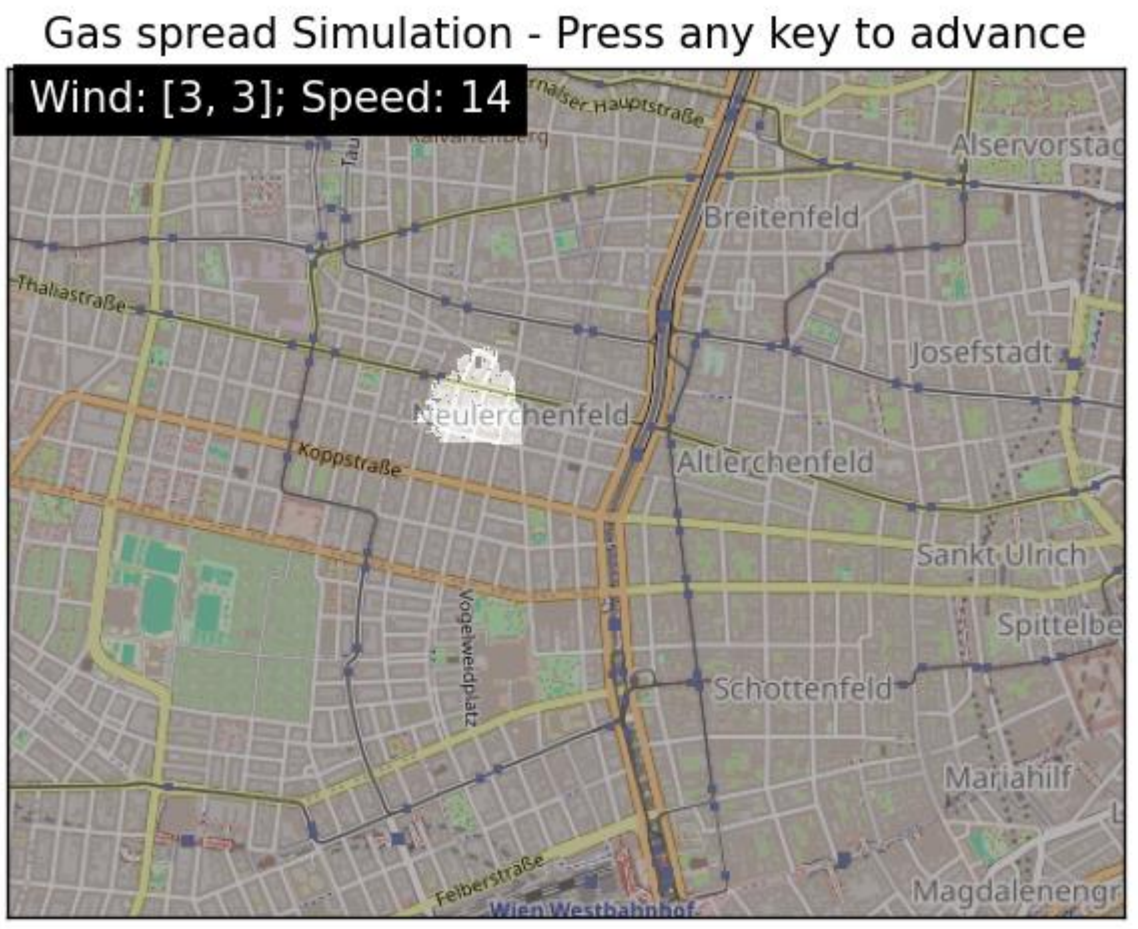


Figure 19 Gas spread simulation

3.6.5.2 *Optimum paths selection*

This is similar to the simulation in case of an earthquake. Please refer to Chapter 3.4 to see how

- Graphs are created by changing data received from OpenStreetMap
- The paths between two points are computed
- The simulation of requests in each node of the graph is simulated
- Auto-adaptive simulation is run (Based on RL methods)

The description of inputs, outputs, algorithms, user interface and REST API from Chapter 3.4 is applicable also here.

For a better understanding, we are presenting a short version of the whole process here.

After entering data for a specific area, the user selects two points: a Start Point and an End Point. The system then calculates the possible paths between these points.

The computation process includes the following steps:

- Cycle Removal – Any cycles in the graph are eliminated to ensure valid paths.
- Shortest Path Calculation – The system identifies the shortest paths between the selected points.
- Independent Path Computation – Multiple independent paths are determined to provide alternative routing options.

If no paths are available, the system displays notifications to inform the user.

The computed results are:

- Displayed on a map for visual interpretation.
- Saved as an HTML file for future reference.

The paths from one selected point to another are visually represented as shown in the following figure. (Figure 20 Best routes to avoid damages):

The REAT APIs used are:

```
@app.post("/compute-shortest-path")
async def api_compute_shortest_path(request: PathRequest):


@app.post("/compute-independent-path")
async def api_compute_independent_path(request: PathRequest):
```

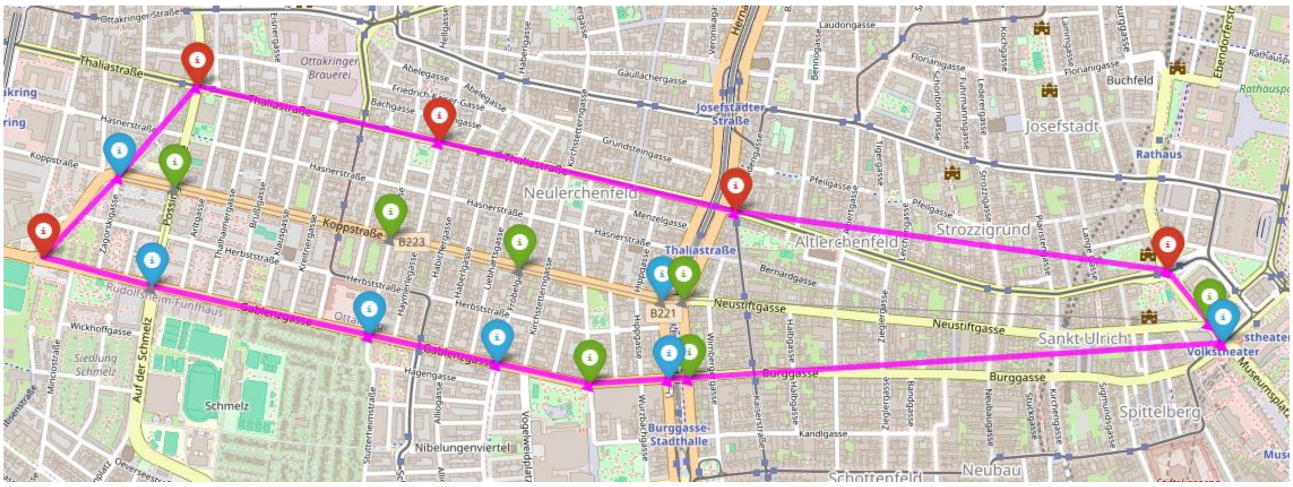


Figure 20 Best routes to avoid damages

3.6.5.3 *RL for Adaptive Resource Deployment*

This is similar to the simulation in the case of a heatwave. Please refer to Chapter 3.5 for a full description of inputs, outputs, algorithm and user interface.

We are presenting here a summary of this process applied to the specific use case.

The final step involves running an auto-adaptive queue simulation. This process first initiates the simulation and then dynamically adjusts the node capacities to reduce queue lengths.

We have developed methods based on Reinforcement Learning (RL). After one of the methods is applied, on the right side, the queue length before adaptation is displayed, showing values between 30 and 70 for all nodes in the paths.

After applying RL, the queue length reduces to a range of 0 to 22, demonstrating a significant improvement in efficiency.

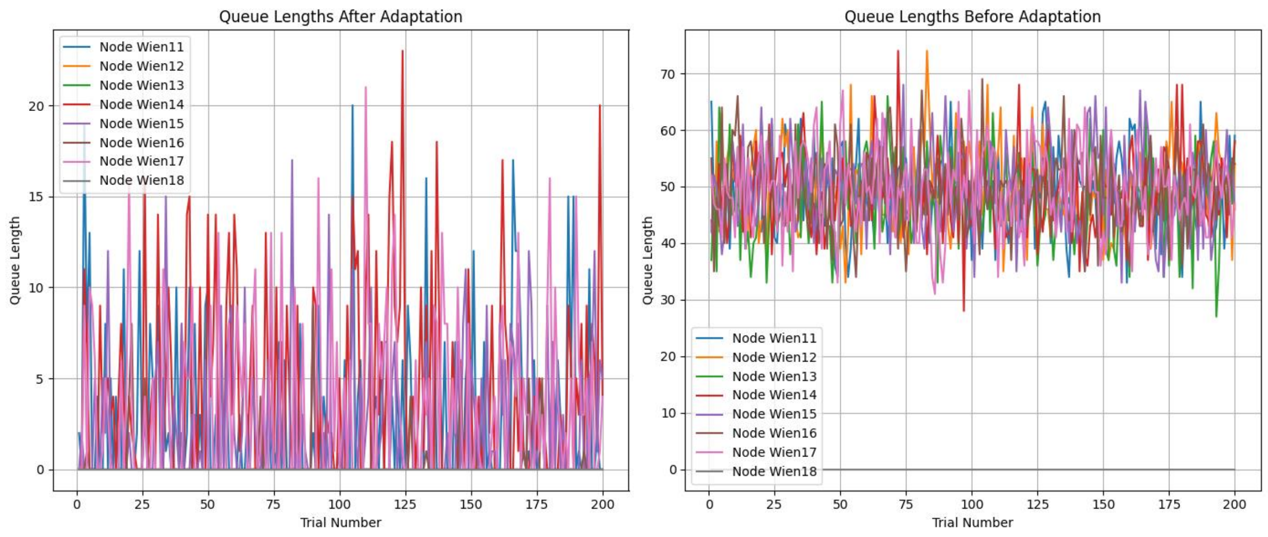


Figure 21 Queue lengths after and before self-adaption

4. CONCLUSIONS AND FUTURE WORK

The **D4.3 – Enhanced Intelligence & Self-adaptive Simulations** deliverable presents a **comprehensive framework** for integrating **Artificial Intelligence (AI) and Machine Learning (ML) algorithms** into **self-adaptive simulation models** for **Community Disaster Risk Management (DRM)**.

The main contributions refer to:

AI-Enhanced Disaster Simulations: The integration of Bayesian optimization, reinforcement learning (RL), and anomaly detection enables simulations to adjust dynamically to environmental changes.

Self-Adaptive Modelling for Different Disaster Scenarios:

- Wildfire models adjust based on wind speed, fuel load, and fire spread patterns to optimize evacuation strategies.
- Earthquake models incorporate seismic data, infrastructure resilience, and traffic patterns to improve disaster response.
- Heatwave models dynamically manage hospital triage, ambulance dispatch, and traffic flow to prevent system overloads.
- Man-made disaster models simulate toxic gas dispersion, industrial accidents, and emergency evacuation logistics for efficient containment and public safety.

Graph Theory and Queue Theory for Optimization:

- Graph-based modelling enhances route planning, evacuation management, and emergency resource allocation by computing shortest and independent paths.
- Queue theory models (M/M/c) simulate traffic congestion, hospital triage, and emergency response bottlenecks, providing adaptive solutions through AI-based decision-making.

Real-Time Processing and Decision Support:

- REST API integrations enable seamless data processing, visualization, and interaction with emergency management platforms.

The current work is intended to be a starting point for future work including:

- **Improving real-time data integration** by incorporating satellite and IoT sensor data for more accurate predictions.
- **Refining AI models with advanced reinforcement learning techniques** to further enhance adaptability.
- **Developing a unified simulation dashboard** for real-time visualization and interactive decision-making.
- **Extending the self-adaptive approach to additional disaster scenarios**, including floods and pandemics.

5. REFERENCES

Anand Deshpande, M. K. (2018). *Artificial Inteligence for Big Data.* Packt.

Apache. (2024). *ApacheMQ*. Retrieved from https://activemq.apache.org/

Apache Airflow. (2024). Retrieved from Apache Airflow: https://www.geeksforgeeks.org/what-is-apache-airflow/

Apache Kafka. (2024). *Apache Kafka*. Retrieved from https://kafka.apache.org/

Apache Kafka vs RabitMQ. (2024). Retrieved from https://www.upsolver.com/blog/kafka-versus-rabbitmq-architecture-performance-use-case

Apache NIFI. (2024). Retrieved from Apache NIFI: https://nifi.apache.org/

Berge, C. (1982). *The Theory of Graphs and Its Applications.* Greenwood Press.

Catherine Forbes, M. E. (2011). *Statistical Distributions - Fourth Edition.* WILEY.

Davison, A. C. (2008). *Statistical Models (Cambridge Series in Statistical and Probabilistic Mathematics, Series Number 11).* Cambridge University Press.

Diestel, R. (2017). *Graph Theory - Fifth Edition.* Springer.

Docker documentation. (2024). Retrieved from Docker documentation: https://docs.docker.com/get-started/overview/

Docker Swarm. (2024). Retrieved from Docker Swarm: https://www.geeksforgeeks.org/introduction-to-docker-swarm-mode/

Donald Gross, J. F. (2008). *Fundamentals of Queueing Theory Fourth edition.* Wiley-ISBN: 9780471791270.

IEEE. (2022). IEEE Recommended Practice for Distributed Simulation Engineering and Execution Process (DSEEP. Retrieved from https://standards.ieee.org/ieee/1730/10715/

IEEE-1730. (2022). *https://ieeexplore.ieee.org/document/9919118.* IEEE.

ISO/IEC/IEEE 24748-6:2023(en). (2023). Retrieved from ISO/IEC/IEEE 24748-6:2023(en): https://www.iso.org/obp/ui/en/#iso:std:iso-iec-ieee:24748:-6:ed-1:v1:en

Keycloack. (2024). *Keycloack*. Retrieved from https://www.keycloak.org/documentation

Kubernetes. (2024). Retrieved from Kubernetes: https://kubernetes.io/

Kubernetes vs Swarm. (2024). Retrieved from Kubernetes vs Swarm: https://phoenixnap.com/blog/kubernetes-vs-docker-swarm

Lapan, M. (2020). *Deep Reinforcement Learning Hands-On (Second Edition).* Packt-ISBN: 9781838826994.

Lee, A. M. (1966). *Applied Queueing Theory.* London: Macmillan Press Limited.

Liu, H. H. (2018). *Machine Learning - A Quantitative Approach.* PerfMath.

Maxwell B. Joseph, M. W. (2019). Spatiotemporal prediction of wildfire size extremes with Bayesian finite sample maxima. *Ecological Aplications, september 2019*, 1266-1281.

Mulesoft Anypoint. (2024). Retrieved from Mulesoft Anypoint: https://www.mulesoft.com/

Ogata, Y. (1988). Statistical Models for Earthquake Occurrences and Residual Analysis for Point Processes. *Journal of the American Statistical Association, 83(401)*, 9-27.

PANTHEON - Consortia. (2023). *DOA.*

Pantheon Consortia. (2024). *D3.6-Use Case Scenarios.*

Pantheon Consortia. (2024). *D3.7 Pantheon System Architecture.*

PANTHEON Consortia. (2024). *D4.2-Conceptual models.*

PMI. (2024). *Agile Practice Guide.* Retrieved from https://www.agilealliance.org/wp-content/uploads/2021/02/AgilePracticeGuide.pdf

RabitMQ. (2024). *RabitMQ*. Retrieved from https://www.rabbitmq.com/

Zannetti, P. (1990). *Air Pollution Modeling-Theories, Computational Methods and Available Software-ISBN 978-1-4757-4467-5.* Springer.