



Critical Action Planning over Extreme-Scale Data

D4.2 Final Report on Complex Event Forecasting, Learning and Analytics Version 1.0

Documentation Information

Contract Number	101092749
Project Website	https://crexdata.eu/
Contractual Deadline	M36, December 2025
Dissemination Level	PU - Public
Nature	R - Report
Author	Elias Alevizos (NCSR)
Contributors	C Akasiadis, E Alevizos, A Artikis, D Banelas, A Deligiannakis, D Özcan, N Döhner, M Ebel, G Grigoropoulos, N Giatrakos, M Juvillà, P Mantenoglou, A Montagud, G Katsimpras, T Ntiniakou, I Martinez, J Orama, G Paliouras, M Ponce de Leon, J Pottebaum, V Samoladas, P Sklavos, M Theologitis
Reviewers	Gennady Andrienko, Nathaliya Andriyenko
Keywords	Complex event recognition/forecasting, optimization, data streams, federated machine learning, multilingual text mining, event detection



CREXDATA has received funding from the European Union's Horizon Europe programme under grant agreement number 101092749.

Change Log

Version	Author	Date	Description Change
V0.1	E Alevizos	16-07-2025	Initial setup
V0.2	All partners	21-11-2025	Contributions from the tasks
V0.3	All partners	27-11-2025	First corrections after internal review
V0.4	All partners	01-12-2025	Final corrections after internal review
V0.5	E Alevizos	05-12-2025	Final editing and typesetting
v1.0	A. Deligiannakis	23-12-2025	Final pass, minor corrections/formatting

Contents

Executive Summary	10
1 Introduction	12
1.1 Purpose and scope	12
1.2 Relation to other Deliverables	12
1.3 Source code availability	12
1.4 Structure of the Deliverable	13
2 Multi-Resolution Complex Event Forecasting	14
2.1 Forecasting with forward recognition	14
2.2 Hierarchical complex event processing	19
2.2.1 Introduction	19
2.2.2 Background and problem formulation	22
2.2.3 Run sharing and synchronization	32
2.2.4 Empirical evaluation	34
2.3 Sequencing in the Run-Time Event Calculus	34
2.4 Components for Critical Maritime Event Forecasting and Resolution	36
2.4.1 MAR_2: Hazardous weather routing	37
2.5 Discussion & Lessons Learnt	41
3 Interactive Learning for Simulation Exploration	46
3.1 Emergency Use-Case	46
3.1.1 Barrier Height Optimization	50
3.1.2 Optimizer performance comparison	51
3.1.3 Workflow Evaluation & Lessons Learnt	54
3.2 Health Crisis Use-Case: Multi-scale Simulations	57
3.2.1 Quick Outcome Characterization via Early Time-series Classification	59
3.2.2 Evaluation	60
3.2.3 Lessons Learned	61
3.3 Health Crisis Use-Case: Epidemiological Scenario	62
3.3.1 Reinforcement Learning	63
3.3.2 Evaluation & Lessons Learned	65
3.4 Maritime Use-Case	67
3.4.1 Interactive Learning Workflow	67
3.4.2 Vessel Collision Avoidance Interactive Simulator	68
3.4.3 Hazardous Weather Routing Interactive Simulator	70
3.4.4 Evaluation & Lessons Learnt	73
4 Federated Machine Learning	75
4.1 Introduction	75
4.2 FDA-OPT: Generalized Federated Dynamic Averaging	75
4.2.1 Default Configuration for FDA-OPT	77
4.2.2 Discussion	78
4.2.3 SKETCHFDA vs. LINEARFDA	79
4.3 Experiments & Analysis for FDA-OPT	80

4.3.1	Setup	80
4.3.2	Results	82
4.4	Component Details - FDA-OPT	85
4.4.1	Standalone Architecture	85
4.4.2	Configurations	86
4.4.3	Integration: With or Without Graphical Interface	87
4.5	NS3: Accurate Network Modeling	87
4.6	An Adaptive NeRF Framework for City-Scale Emergency Awareness	90
4.6.1	Foundational Methods and System Components	91
4.6.2	Pipeline Overview	91
4.6.3	Offline Stage: Meta Learning	93
4.6.4	Online Stage: Runtime Adaptation	97
4.6.5	Rendering and Use Case Integration	99
4.7	Uncertainty-aware Synchronization Protocols	100
4.8	Lessons Learned	101
5	Optimized Distributed Analytics as a Service	103
5.1	Overview of Optimization Approaches in CREXDATA	103
5.2	Recap on APEROL's Problem Setup	103
5.2.1	Logical Workflows vs. Physical Plans	104
5.2.2	Algorithmic Exploration of the Graph of Plans	105
5.3	Introducing ESQ and Heuristic in the APEROL Suite	105
5.4	The DAG* Algorithm for Optimal Physical Plans	107
5.4.1	DAG* New Problem Formulation	107
5.4.2	The Proposed DAG* Algorithm	110
5.4.3	On DAG* Optimality	113
5.4.4	Dynamic Setting	114
5.5	Experimental Evaluation	116
5.5.1	Objectives, Metrics and Workload	116
5.5.2	Real Testbed Configuration	118
5.5.3	Simulated Networks Configuration	118
5.5.4	Algorithmic Setup & Hardware	119
5.5.5	APEROL Standalone Evaluation on Simulated Setups	121
5.5.6	APEROL - DAG vs SOTA: Comparative Evaluation on Simulated Networks and Real Testbed	124
5.5.7	Scalability for Extreme Scale Analytics	125
5.6	Optimization Lessons Learned	128
6	Text mining for Event Extraction	129
6.1	Introduction	129
6.2	System Architecture	129
6.2.1	Overview	129
6.2.2	Development Methodology	130
6.3	Crawler Component	131
6.4	Event-type Prediction Component	133
6.4.1	Model Design Decisions	133
6.4.2	Improving prediction performance	137

6.4.3	Integration	140
6.5	Question Answering Component	140
6.5.1	Extended Evaluation	142
6.5.2	Post-Retrieval Clustering	144
6.5.3	Integration	146
6.6	Graphical User Interface	146
6.6.1	Side Bar	147
6.6.2	Crawler UI Tab	148
6.6.3	Relevance UI Tab	149
6.6.4	Question Answering UI Tab	149
6.6.5	DEBUG/LOG Console Tab	151
6.7	Lessons Learned	152
7	Conclusions	154
8	Acronyms and Abbreviations	157

List of Figures

Figure 1	Example <i>SRT</i> .	14
Figure 2	Wayeb forecasting with probabilistic model.	16
Figure 3	Wayeb forecasting with forward recognition.	17
Figure 4	Water and rain sensors in Innsbruck.	17
Figure 5	Time series for sensor KA0858.	18
Figure 6	ARIMA, resolution = 30 minutes, horizon = 1, 7.	18
Figure 7	LSTM, resolution = 30 minutes, horizon = 1, 7.	19
Figure 8	KPIs for task 4.1.	19
Figure 9	KA0163 for resolution of 30 minutes.	20
Figure 10	TFT results for multiple sensors for a resolution of 30 minutes.	20
Figure 11	Experimental results.	35
Figure 12	Weather routing architecture overview	43
Figure 13	Example vessel traffic patterns based on historical wave heights (low: <2m, moderate: 2-4m, high: >4m).	44
Figure 14	Routing graph around the Florida peninsula. Purple: H3 nodes; green: valid 1-step edges; blue: discarded multi-step transitions; brown: masked edges excluded by sea mask.	44
Figure 15	Example of the weather routing application output on the user interface.	45
Figure 16	Weather routing request latency [ms] as a function route length [km].	45
Figure 17	Conceptual workflow of the Emergency use-case interactive learning scenario for flood control.	46
Figure 18	AI Studio workflow that implements the Emergency use-case interactive learning scenario for flood control.	48
Figure 19	Kafka topic streams of the barrier height optimization.	50
Figure 20	AI Studio workflow overview.	50
Figure 21	AI Studio workflow subprocesses.	50
Figure 22	AI Studio workflow subprocesses.	51
Figure 23	Linear objective function and championed points by each optimizer case.	52
Figure 24	Non-linear objective function and championed points by each optimizer case.	52
Figure 25	Initial ARGOS view.	54
Figure 26	Opening the Barrier Menu.	55
Figure 27	Setting a barrier location.	55
Figure 28	Setting the Point of Interest.	56
Figure 29	Setting an additional barrier location.	56
Figure 30	Defining simulation details.	56
Figure 31	Browser-based dashboard used in the multiscale interactive learning workflow. The interface combines real-time cell-state trajectories, early ETSC predictions, and an intervention panel that allows the user to trigger drug or gene knock-outs during execution.	58

Figure 32	Human-in-the-loop intervention workflow in RapidMiner. The process connects the Kafka Cluster (input data stream) to a Python-based intervention node and a Write-Kafka operator that publishes JSON control messages (pb_drug). These updates are consumed by the PhysiBoSS orchestrator while a browser-based dashboard visualises real-time cell counts, enabling experts to steer infection dynamics interactively.	59
Figure 33	Overview of the interactive learning workflow coupled with episim-emews. The figure illustrates the main steps of the episim-emews workflow along with the interactive learning process enabled by user-driven analysis between successive experiments	62
Figure 34	Diagram of the episim-rl workflow, illustrating the integration of reinforcement learning techniques into EpiSim.jl. The process begins by restarting the simulation and updating the configuration file with the new NPI parameters (applied policy). EpiSim.jl is then executed, the outputs are collected and sent to the RL module, which updates the agent and selects a new action, restarting the cycle	64
Figure 35	Maps of the root mean square error (RMSE) across provinces for the first and last experiments, illustrating the improvement of the calibration. . . .	66
Figure 36	A): Convergence of the total reward over the training episodes for the three scenarios. B): Top episode from each scenario, showing at the top the confinement policies applied every two weeks and, at the bottom, displaying the simulated curves for cases, hospitalizations, and deaths. These are compared with the calibrated simulation that represents the real outcome, illustrating improved results.	67
Figure 37	Collision avoidance simulator interface. The display shows two vessels approaching a forecasted collision point, the computed collision-avoidance path (green), the collision forecast location (purple marker), and an overlay of vessel traffic flow density. The control panel allows the operator to set the target speed and reroute.	69
Figure 38	Output of the speed-based simulation function. In this example the operator specifies target speeds of 15 kn (red) and 10 kn (blue) for the two vessels as the forecasted collision positions lies too far ahead. The simulator computes new collision-avoidance paths (yellow-coloured paths) that divert the vessels around the forecasted collision point.	70
Figure 39	Hazardous weather routing simulator interface using the updated labeled screenshot. Weather risk is visualized as shaded regions over the Western Mediterranean. The interface distinguishes between a traffic-optimised route (blue dashed line), a weather-optimised route (green solid line) and the shortest route (brown dotted line). Operators can choose among dedicated route generators (traffic, shortest) and simulation functions for departure time and speed, define the origin and destination ports, and select weather forecast layers. The timeline at the bottom enables exploration of the temporal evolution of the forecast along each route, and numeric markers indicate the severity of weather events in each sector.	72

Figure 40	Hazardous weather routing simulator interface on Pekko processed AIS positions. Users are able to select a vessel based on its last reported AIS position from the MarineTraffic AIS stream and view its projected weather route towards its reported destination port in real-time. Users may also directly access the simulator functions for that vessel directly through the simulator window to evaluate alternative travel speeds and routes.	73
Figure 41	SKETCHFDA & LINEARFDA: Local State structure.	79
Figure 42	Training Loss progression of training RoBERTa with FDA-OPT vs. FEDOPT on MRPC (left) and SST-2 (right)	84
Figure 43	The <i>mediator</i> acts as a persistent orchestrator that continuously monitors a dedicated <i>kafka</i> topic for incoming FL job requests, which are transmitted in the form of structured JSON messages. Upon receiving such a job specification, the mediator proceeds to parse the configuration and automatically deploy a Federated Learning (FL) cluster tailored to the received settings	86
Figure 44	Upon receiving a simulation request, the <i>mediator</i> spawns an NS3-Runner to execute a single NS-3 job. Results are retrieved from cache when available, or generated through parallel round simulations. In the presented example we assume the first k rounds are cached while the rest are simulated.	88
Figure 45	Overview of the proposed city-scale NeRF infrastructure. Drones collect baseline imagery under normal conditions to pretrain city-scale NeRF models. The environment is divided into spatial modules, each optimized via meta-continual learning for rapid adaptation. During emergencies, new drone imagery from affected areas is streamed into the corresponding modules, enabling fast NeRF updates and providing first responders with up-to-date 3D visualizations for situational awareness.	92
Figure 46	Example of meta-learned NeRF initialization for a single spatial region. The real scene (left) and the corresponding meta-learned representation (right) show that the model learns a coarse, structured prior rather than a fully detailed reconstruction	93
Figure 47	Viewer instance demonstrating the interactive control API. On the left we have the rendered view and the right the controls for seamless user interaction.	99
Figure 48	Example of DAG* output on the STATS workflow [69]. Nodes are the operators of the analytics workflow. Disks in each node are possible configurations, i.e., executing the operator represented by Node i on network device d_k . Each disk has a different cost (processing latency). An edge denotes operator dependency with a different cost weight (network latency) depending on the configurations they connect. Green-colored edges and disks compose the optimal graph returned by DAG*.	109
Figure 49	ETL Workflow	117
Figure 50	PRED Workflow	117
Figure 51	TRAIN Workflow	117
Figure 52	STATS Workflow	117
Figure 53	YAHOO Workflow	117
Figure 54	Visual illustration of the Real Testbed	119
Figure 55	Visual illustration of the simulated networks that are used in the experimental evaluation	120

Figure 56	Heterogeneous Setup – ETL Workflow: APEROL Plan Cost Improvement (higher is better) across network sizes vs. number of adaptation decisions.	121
Figure 57	ETL: Average time per adaptation decision.	121
Figure 58	Heterogeneous Setup – PRED Workflows: APEROL Plan Cost Improvement (higher is better) across network sizes vs. number of adaptation decisions.	122
Figure 59	PRED: Average time per adaptation decision.	122
Figure 60	Heterogeneous Setup – STATS Workflow: APEROL Plan Cost Improvement (higher is better) across network sizes vs. number of adaptation decisions.	122
Figure 61	STATS: Average time per adaptation decision.	122
Figure 62	Heterogeneous Setup – TRAIN Workflow: APEROL Plan Cost Improvement (higher is better) across network sizes vs. number of adaptation decisions.	123
Figure 63	TRAIN: Average time per adaptation decision.	123
Figure 64	Homogeneous-ETL: Plan cost and execution time.	124
Figure 65	Homogeneous-STATS: Plan cost and execution time.	124
Figure 66	APEROL, DAG* vs. competitors on synthetic (PRED, TRAIN) and real (YAHOO) workloads. Each row shows plan latency (left) and execution time (right).	126
Figure 68	High-level system architecture of social media toolkit	130
Figure 69	Crawler Component	131
Figure 70	Sample start and kill JSON crawl commands.	132
Figure 71	Sample Twitter/X message saved to kafka cluster.	133
Figure 72	Zero-shot annotation prompt.	136
Figure 73	Keywords for synthetic data generation.	139
Figure 74	Stream processing latency charts.	141
Figure 75	RapidMiner workflow for training BERT models.	141
Figure 76	Impact of the number of passages.	143
Figure 77	Impact of the number of relevant documents.	143
Figure 78	Evaluation of different rank aggregation methods within the QA pipeline. The scores represent the results of each model using BM25 for retrieval (scores in parentheses correspond to the results using Contriever).	144
Figure 79	Average time (seconds) required for processing a query.	144
Figure 80	Post-retrieval clustering.	145
Figure 81	Question answering component as RapidMiner endpoint.	146
Figure 82	Text mining GUI: Side bar	147
Figure 83	Text mining GUI: Crawl UI tab	148
Figure 84	Text mining GUI: Relevance UI tab	149
Figure 85	Text mining GUI: Question Answering tab	150
Figure 86	Sample no-RAG response in question answering process	151
Figure 87	Sample Genra-RAG repsonse in question answering process.	151
Figure 88	Text mining GUI: Debug and Log console tab	152

Executive Summary

This document presents the final advancements and discusses the scientific and technological issues that are being investigated in Work Package 4, with respect to Complex Event Forecasting, Learning and Analytics.

First, it presents a method for multi-resolution Complex Event Forecasting based on forward recognition. This method of forecasting via forward recognition offers several advantages:

- We can run various time series forecasting methods, depending on the dataset and the domain. In some cases, a simple ARIMA might be enough, whereas in others we might need a time series foundation model.
- We can perform time series aggregation and/or smoothing so that we can work at various resolutions.
- We can run forward recognition for multiple values of window center and size.
- For each method, we can estimate the Volume under Curve (VUC) as a forecasting metric.

It also presents our work on hierarchical Complex Event Recognition. Typically, hierarchical Complex Event Recognition relies on an event passing mechanism, which results in several soundness issues. We show how this problem can be avoided, using a top-down synchronization method. Such a hierarchical system could then be used to perform forecasting at multiple (resolution) levels.

We then present the interactive learning approaches that have been developed for each use-case of CREXDATA, and integrated into respective AI Studio workflows. For the Emergency use-case, we devised a workflow that assists first line responders in flooding scenarios. From the perspective of WP4, an optimizer has been developed that optimizes the height of water barriers placed on a map to keep a particular area of interest (e.g. a hospital, an airport, etc.) dry and safe. For the Health crisis use-case, we developed and integrated an Early Time Series Classification algorithm to detect uninteresting multi-scale simulations before their completion and terminate them at early stages, thus saving time and computational resources. Also, a Reinforcement Learning agent has been employed to optimize the health intervention policies in face of epidemics. Finally, for the Maritime use-case, we have put forward solutions for vessel collision forecasting and rerouting and for the hazardous weather rerouting problems.

Our work on Federated Machine Learning, presented next, focused on developing a novel, auto-tuned, variation of our Functional Dynamic Averaging algorithm, for non-expert users. We also integrate the NS3 network simulator with our federated learning algorithms, in order to allow the user to estimate the impact of different parameter settings. Finally, we develop a novel, scalable, and efficient NeRF technique for rapidly building an easy to visualize 3D model of an emergency area from images by distributed drones.

Our work on Optimized Distributed Analytics as a Service, presented next, presents CREXDATA's optimizer for distributing complex analytics workflows across a cloud-edge continuum, built around the APEROL suite of parallel search algorithms and the DAG* optimal planner. APEROL models the problem as a Graph of Plans, where each node is a full physical

workflow (operator-device/platform assignments) and edges correspond to atomic placement changes; it then explores this combinatorial space under a multi-objective cost model that jointly optimizes throughput, end-to-end latency, communication cost, and migration overhead. Beyond the initial Exhaustive Search with Counting (ESC), Random Sampling Search (RSS) and Greedy algorithms, the chapter introduces two new variants - Exhaustive Search with Queue (ESQ) and a Pareto-front-guided HEURISTIC search - that steer exploration toward high-quality plans while exploiting parallel workers. In contrast, DAG* reformulates optimization as an A*-like search over partial assignments in a DAG, using a novel admissible heuristic to prune most candidate graphs and guarantee the globally optimal plan with respect to latency, together with an incremental variant (d-DAG*) for reacting to dynamic cost changes. An extensive evaluation on both simulated networks (up to 2047 devices) and a 194-node real testbed with Yahoo and RIoT benchmark workflows shows that APEROL and DAG* are complementary and highly scalable to extreme scale analytics. APEROL scales exhaustive search to over 2M plans per second on commodity hardware. DAG* can timely devise the optimal plan pruning the search space by two orders of magnitude in adversarial setups and up to 3 orders of magnitude in realistic settings. APEROL and DAG* outperform the state-of-the-art providing execution plans of, on average $5\times$ and, up to $10\times$ better end-to-end latency.

Finally, we present a social media toolkit developed to crawl multilingual social media posts, extract information, and integrate it into CREXDATA. The toolkit includes a crawler component for collecting posts as they are published, a multilingual language model for low-latency event type prediction, and a question answering component for extracting information from relevant posts. In addition, we developed a text mining graphical user interface that brings all components together into a single, unified workflow.

1 Introduction

This document presents the final achievements of the CREXDATA project with respect to Complex Event Forecasting, Learning and Analytics.

1.1 Purpose and scope

The reader is expected to be familiar with Complex Event Forecasting, Artificial Intelligence, Federated Learning, Text Mining and Distributed processing techniques, as well as the general intent and concept of the CREXDATA project. The target readership is:

- CREXDATA researchers;
- CREXDATA audit.

CREXDATA focuses on event forecasting, interactive and federated machine learning and text mining techniques for large scale data. This document presents the current advancements and discusses the scientific and technological issues that are being investigated in Work-Package 4, with respect to Complex Event Forecasting, Learning and Analytics.

1.2 Relation to other Deliverables

This document is related to the following project deliverables:

- D2.2 Initial Use Case Evaluation, Pilots, Demonstrators and Simulation Models and Tools;
- D2.3 Final Use Case Evaluation, Pilots, Demonstrators and Simulation Models and Tools;
- D3.1 Initial Report on System Architecture, Integration and Released Software Stacks;
- D4.1 Initial Report on Complex Event Forecasting, Learning and Analytics;
- D3.2 Final Report on System Architecture, Integration and Released Software Stacks.

1.3 Source code availability

CREXDATA has promised to deliver the source code of its solutions. Below we provide a list of source code repositories related to the present deliverable. Please note that some of the repositories may be anonymous because the relevant papers have not been published yet.

- Repositories for Task 4.1
 - [https://github.com/altairengineering/crexdata-public/tree/main/CREXDATA%20System%20Components/Complex%20Event%20Forecasting%20\(CEF\)/Complex%20Event%20Forecasting%20\(CEF\)/Wayeb](https://github.com/altairengineering/crexdata-public/tree/main/CREXDATA%20System%20Components/Complex%20Event%20Forecasting%20(CEF)/Complex%20Event%20Forecasting%20(CEF)/Wayeb)
 - [https://github.com/altairengineering/crexdata-public/tree/main/CREXDATA%20System%20Components/Complex%20Event%20Forecasting%20\(CEF\)/Complex%20Event%20Forecasting%20\(CEF\)/RTCEF](https://github.com/altairengineering/crexdata-public/tree/main/CREXDATA%20System%20Components/Complex%20Event%20Forecasting%20(CEF)/Complex%20Event%20Forecasting%20(CEF)/RTCEF)

- Repositories for Task 4.2
 - [https://github.com/altairengineering/crexdata-public/tree/main/CREXDATA%20System%20Components/Complex%20Event%20Forecasting%20\(CEF\)/Complex%20Event%20Forecasting%20\(CEF\)/ETSC](https://github.com/altairengineering/crexdata-public/tree/main/CREXDATA%20System%20Components/Complex%20Event%20Forecasting%20(CEF)/Complex%20Event%20Forecasting%20(CEF)/ETSC)
- Repositories for Task 4.3
 - [https://github.com/altairengineering/crexdata-public/tree/main/CREXDATA%20System%20Components/Federated%20Learning%20\(TUC\)](https://github.com/altairengineering/crexdata-public/tree/main/CREXDATA%20System%20Components/Federated%20Learning%20(TUC))
 - [https://github.com/altairengineering/crexdata-public/tree/main/CREXDATA%20System%20Components/NeRF%20Learning%20and%20Visualization%20\(TUC\)](https://github.com/altairengineering/crexdata-public/tree/main/CREXDATA%20System%20Components/NeRF%20Learning%20and%20Visualization%20(TUC))
- Repositories for Task 4.4
 - [https://github.com/altairengineering/crexdata-public/tree/main/CREXDATA%20System%20Components/Cloud-to-Edge%20Optimizer%20\(TUC\)](https://github.com/altairengineering/crexdata-public/tree/main/CREXDATA%20System%20Components/Cloud-to-Edge%20Optimizer%20(TUC))
- Repositories for Task 4.5
 - <https://github.com/altairengineering/crexdata-public/tree/main/CREXDATA%20System%20Components/Text%20Analytics/App%20%3A%20GUI/Text-Mining-UI>
 - <https://github.com/altairengineering/crexdata-public/tree/main/CREXDATA%20System%20Components/Text%20Analytics/RapidMiner%20Customer%20Operators%20%26%20Extensions>
 - <https://github.com/altairengineering/crexdata-public/tree/main/CREXDATA%20System%20Components/Text%20Analytics/Text%20Analytics%20Component>
 - <https://github.com/altairengineering/crexdata-public/tree/main/CREXDATA%20System%20Components/Text%20Analytics/Web%20Crawler%20Component/X-Crawler>

1.4 Structure of the Deliverable

This document has the following structure:

- Section 2 presents our work on Multi-Resolution Complex Event Forecasting.
- Section 3 presents our work on Interactive Learning for Simulation Exploration.
- Section 4 presents our work on Federated Machine Learning.
- Section 5 presents our work on Optimized Distributed Analytics as a Service.
- Section 6 presents our work on Text mining for Event Extraction.

2 Multi-Resolution Complex Event Forecasting

In this Section, we present our work on multi-resolution Complex Event Forecasting (CEF). In deliverable D4.1, we focused mainly on the expressivity and adaptability of CEF. In this deliverable, we explicitly focus on CEF at multiple resolutions. We first describe how we can leverage already existing time series forecasting techniques in order to perform multi-resolution CEF (Section 2.1). Subsequently, we discuss how we can perform sound and efficient hierarchical Complex Event Recognition (CER) (see Section 2.2). The idea is that, by using hierarchies, we could define patterns at multiple resolutions, with those at the lower levels being the most fine-grained. However, since expressive and sound hierarchies are still an open issue in CER, we present our approach to hierarchical CER as a first step towards multi-resolution CEF. We finally briefly discuss how another hierarchical system for CER can be extended to accommodate more expressive patterns (see Section 2.3)

2.1 Forecasting with forward recognition

In D4.1 (also [3]), we described an extension of our CEF engine, called Wayeb. Wayeb is a formal, compositional framework for CER/F based on symbolic automata.¹

Our work addressed the following limitations of Wayeb:

- **Expressivity:** No constraints between participating events were possible. Thus, we could not express important classes of patterns, e.g., trends or aggregates.
- **Explainability:** Wayeb was unable to mark input events as participating in a complex event because it was based on deterministic automata, which cannot distinguish between different runs.
- **Performance:** However, with non-deterministic automata, performance drops significantly due to multiple runs.

We introduced a new computation model, called **symbolic register transducers (SRT)**, which addressed the above limitations.

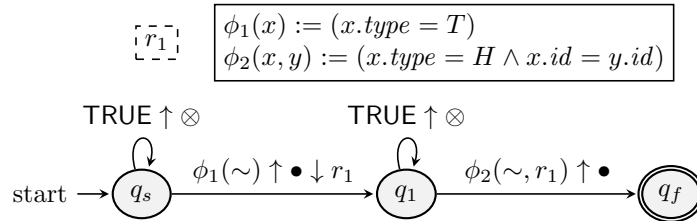


Figure 1: Example *SRT*.

Figure 1 shows an example of a *SRT*. Table 3 shows an example stream. If we apply this *SRT* on this stream, then, after index 4, we would get the following matches:

- $[q_s, \emptyset]$

¹<https://github.com/altairengineering/crexdata-public/tree/main/CREXDATA%20System%20Components/Complex%20Event%20Forecasting%20%28CEF%29/Complex%20Event%20Forecasting%20%28CEF%29/Wayeb>

Table 3: Example stream.

type	T	T	T	H	H	T	...
id	1	1	2	1	1	2	...
value	22	24	32	70	68	33	...
index	1	2	3	4	5	6	...

- $[q_1, \{1\}]$
- $[q_1, \{2\}]$
- $[q_1, \{3\}]$
- $[q_2, \{1, 4\}]$, which is a full match.
- $[q_2, \{2, 4\}]$, which is a full match.

We were able to show the following properties for the language we use to express patterns, called symbolic regular expressions with memory and output (*SREMO*) and *SRT*.

- For every *SREMO* there exists an equivalent *SRT*.
- *SRT* are closed under union, intersection, concatenation and Kleene-star, but not under complement.
- For every windowed *SREMO* there exists an equivalent output-agnostic deterministic *SRT*.
- Output-agnostic *SRT* compiled from windowed *SREMO* are closed under complement.

Typically, CEF in Wayeb works as follows [2]. A pattern is first compiled into an automaton. Then, a training dataset is used in order to construct a probabilistic model for that automaton. This model can then be used to estimate, at runtime, when an automaton is expected to reach one of its final states and, thus, detect a complex event. If the probability of reaching a final state in a given window of input events exceeds some threshold, then a positive forecast is produced for that window. The process is depicted in Figure 2.

One important limitation of this approach is that it fails to scale with large “dictionaries”, i.e., if there are multiple features that the patterns and the probabilistic model need to use. This problem is exacerbated when input events contain multiple variables and when symbolic register transducers are used, where we need to relate current with past events. The size of the “dictionary” and the possible combinations of its “tokens” to relate current and past events can lead to intractable models.

We present an alternative, neuro-symbolic approach. We can first use a neural network to perform raw regression forecasting on the input stream. One advantage of existing neural approaches for time series forecasting is that they can work with multiple variables and resolutions. Instead of forcing Wayeb to generate a probabilistic model, we let a neural model take over this task. We can thus generate a single stream of future input events. We can

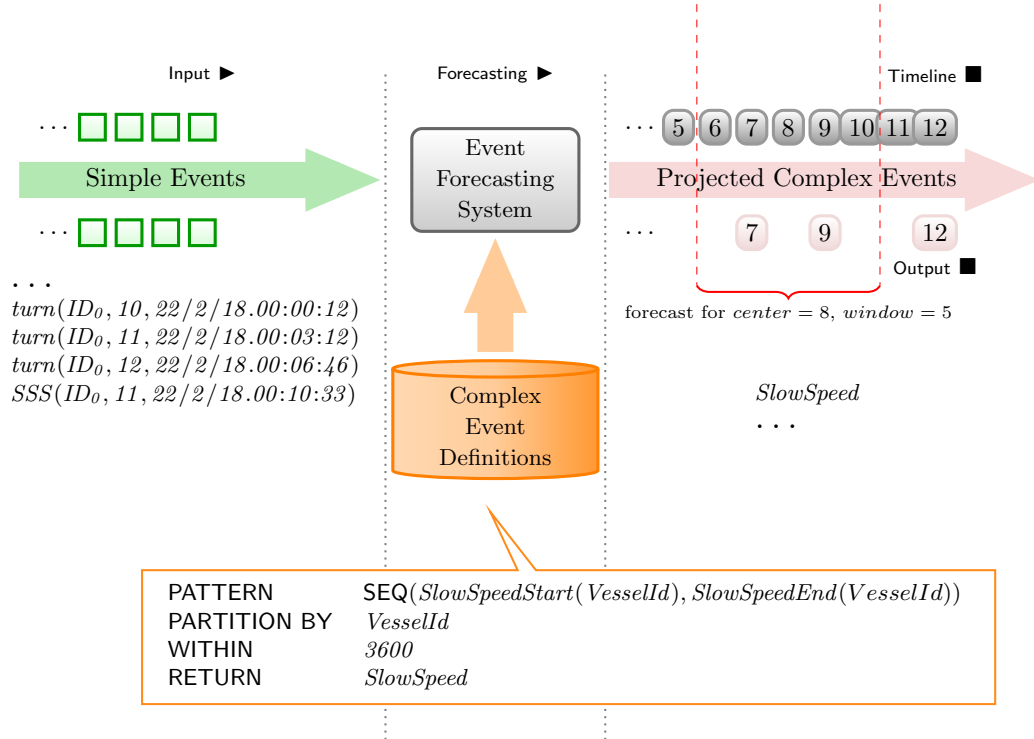


Figure 2: Wayeb forecasting with probabilistic model.

then use the recognition part of Wayeb on top of the forecast events and perform forward recognition on them. With this approach, we can then use the full expressive power of Wayeb patterns, even patterns with registers.

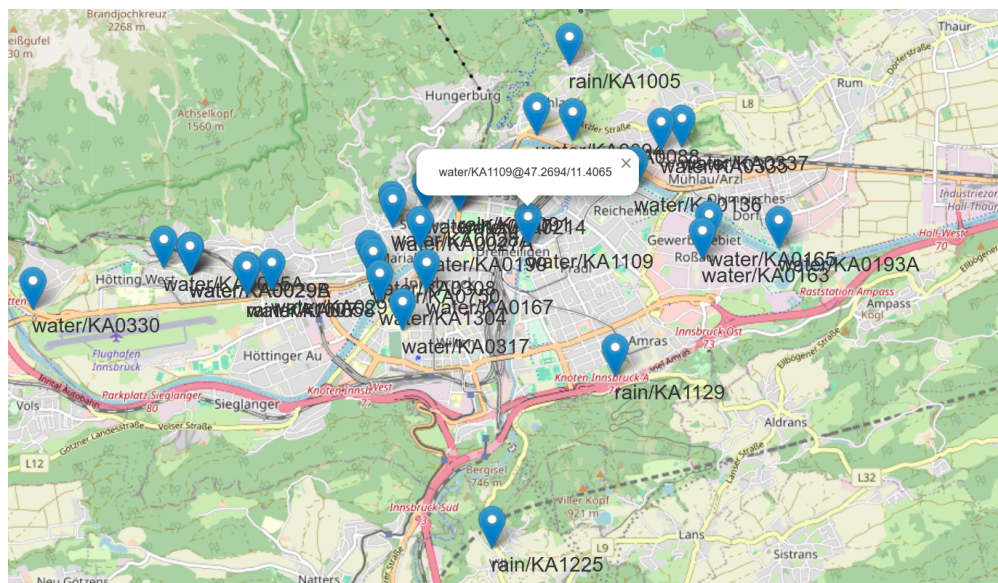
The process in more detail is the following (see Fig. 3). Wayeb typically works with automaton runs that perform recognition. At each timepoint, we initially perform recognition as usual. We then freeze all partial matches of all active runs in the system. A neural method is then used to perform time series forecasting and create a future sequence of events up to some horizon h . The frozen matches can then perform recognition on the future sequence. A set of shadow / ghost matches is thus created. We then check if any of the shadow matches reach a final state within the designated window. If the answer is positive, then we emit a relative positive forecast. Finally, we expunge all shadow matches and unfreeze the frozen partial matches. The same process is repeated for the next timepoint.

This method of forecasting via forward recognition offers several advantages:

- We can run various time series forecasting methods, depending on the dataset and the domain. In some cases, a simple ARIMA might be enough, whereas in others we might need a time series foundation model.
- We can perform time series aggregation and/or smoothing so that we can work at various resolutions.
- We can run forward recognition for multiple values of window center and size.
- For each method, we can estimate the Volume under Curve (VUC) as a forecasting



We applied this method on the Weather Emergency use case. We used a dataset consisting of water level and rain measurements around the city of Innsbruck, spanning 3 years, 2021, 2022 and 2023 (see Fig. 4). Initially, we performed some data synchronization, cleaning



and smoothing. Figure 5 shows one such time series. We then applied piecewise aggregate approximation in order to create versions of the original time series at multiple resolutions. We chose the following resolutions: 0.5 hour, 1 hour, 2 hours, 6 hours, 1 day. For each resolution, we tested the following time series forecasting methods:

- 17

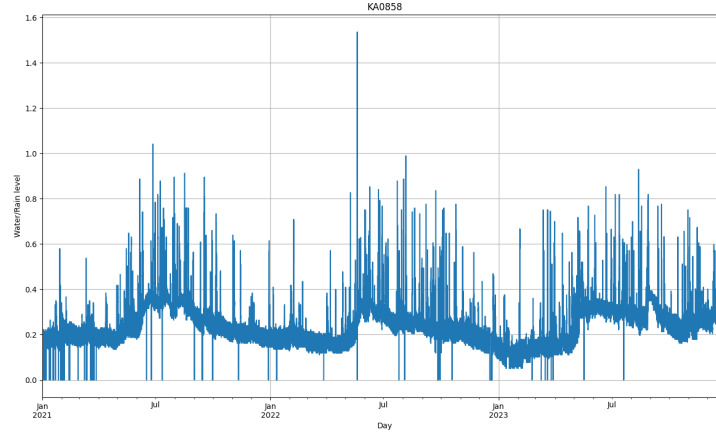
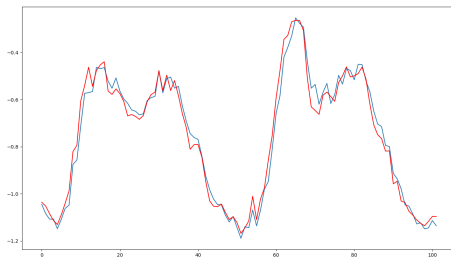


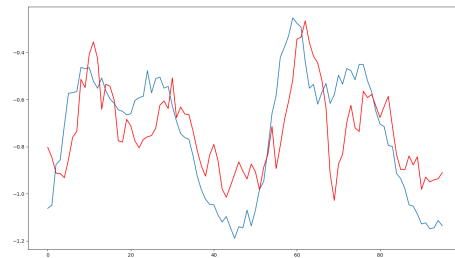
Figure 5: Time series for sensor KA0858.

- LSTM (purely neural, with hyper-parameter search).
- Temporal fusion transformer (TFT), a time series foundation model [41].

Figures 6 and 7 depict some indicative results for ARIMA and LSTM respectively, for two horizons $horizon = 1$ and $horizon = 7$. The blue line shows the actual time series, whereas the red one the predicted values.



(a) $Horizon = 1$.



(b) $Horizon = 7$.

Figure 6: ARIMA, resolution = 30 minutes, horizon = 1, 7.

We then used the predicted time series at each timepoint to perform forward event recognition for each resolution. In this use case, we are interested in forecasting whether the manhole at some point will overflow. We translated this user requirement into several patterns, one for each sensor. According to the patterns, the goal is to detect whether the water level at some sensor will exceed some threshold for multiple consecutive events. The specific threshold is sensor dependent. We applied forward recognition with these patterns on the predicted time series.

Figure 9 shows the relevant accuracy results for one sensor (KA0163) and for a resolution level of 30 minutes. This means that each point on the x and y axes correspond to 30 minutes.

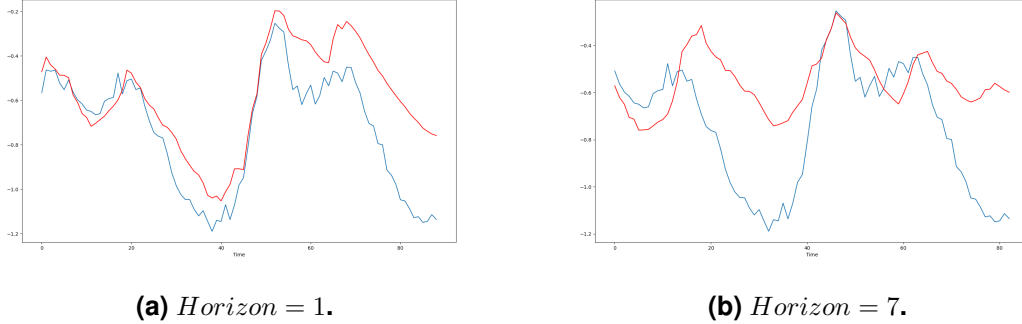


Figure 7: LSTM, resolution = 30 minutes, horizon = 1, 7.

We can see that the TFT method seems to outperform the others, although even a basic ARIMA performs reasonably well. The accuracy tends to drop as the center is increased and the window is decreased. This is to be expected, since an increased window center implies that the target of the forecast is deeper into the future. A decreased window size means that the forecast is more focused on a narrower future range. In both cases, forecasting becomes more difficult. We also show results for other sensors in Figure 10. We depict results only for TFT.

Figure 8 shows the KPIs for Task 4.1. According to the above presented results, we can state that we have fully achieved the objectives of this task. We can successfully forecast critical events from multiple sensors with an accuracy above 75% at least 30 minutes ahead of time. Note that each point in Figure 10 corresponds to 30 minutes of actual time. For example, KA1109 seems to be the most demanding sensor. Even in this case, we can predict overflows 30 minutes ahead of time with an accuracy of 80%.

[R5]: Multiresolution Complex Event Forecasting Component	SC: 3 V: ~7M €	Forecasting >5 critical business events 60 to 15min ahead of time with 75 to 95% accuracy
---	-------------------	---

Figure 8: KPIs for task 4.1.

2.2 Hierarchical complex event processing

2.2.1 Introduction

A Complex Event Recognition (CER) system processes a stream of “simple events” alongside a set of patterns that define relationships between these events, identifying when patterns are satisfied and producing a corresponding stream of “complex events” [26, 12]. Events are typically structured as tuples of numerical or categorical values. Since time is crucial for CER, a temporal formalism is employed to define the patterns to be detected. These patterns impose temporal (and sometimes atemporal) constraints on the input events, and when these constraints are met, a complex event is recognized. Atemporal constraints may be “local”,

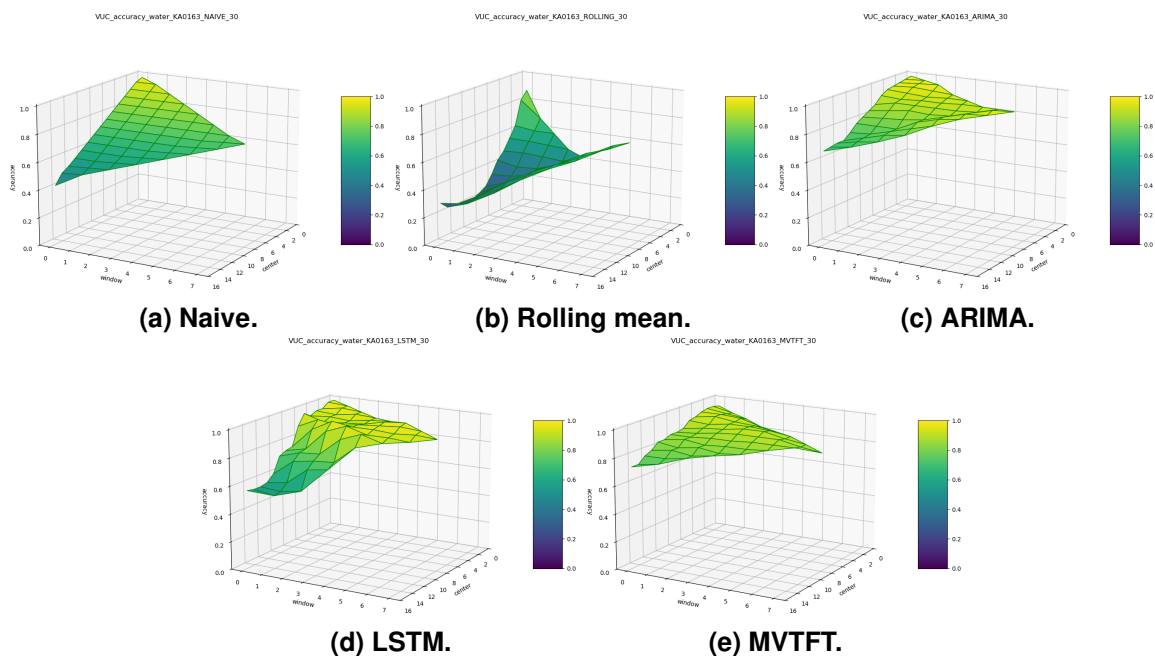


Figure 9: KA0163 for resolution of 30 minutes.

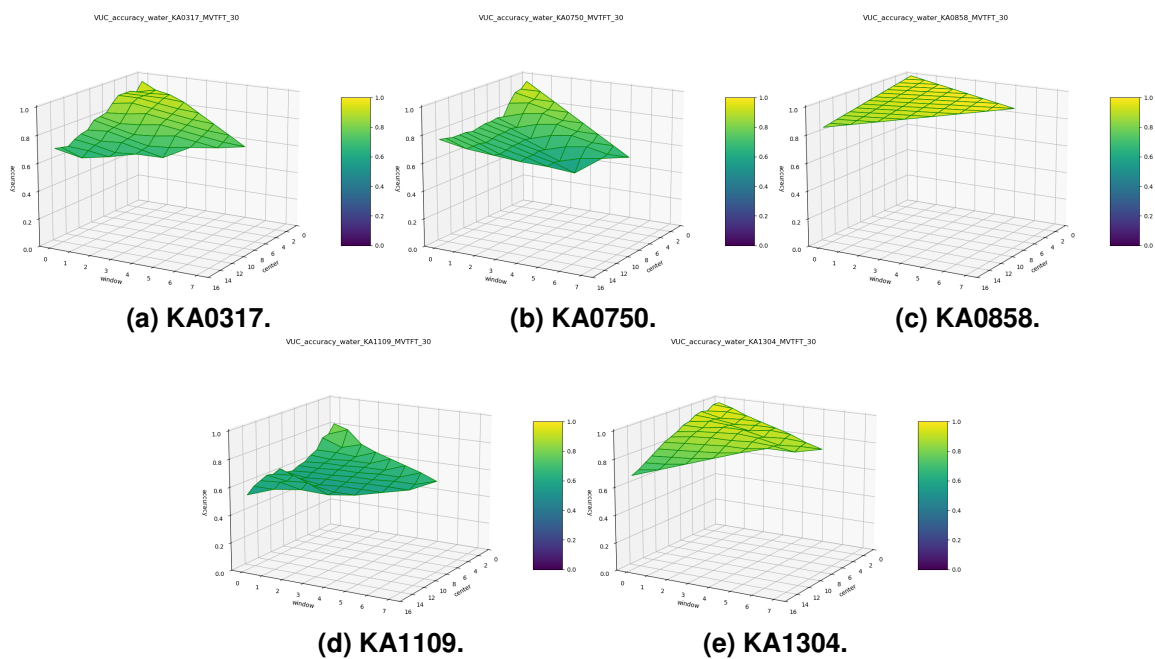


Figure 10: TFT results for multiple sensors for a resolution of 30 minutes.

applying only to the most recent event in the input stream. For instance, in temperature sensor streams, a local constraint could specify that the temperature of the latest event exceeds a certain threshold. More frequently, constraints involve multiple events in the pattern, such as requiring the most recent event's temperature to be higher than that of the previous event.

Automata are particularly relevant to the field of CER because they offer a natural approach to handling event sequences. As a result, traditional regular expression operators, such as concatenation, union, and Kleene star, are often assigned an implicit temporal interpretation in CER. For instance, the concatenation of two events is considered to occur when the second event is processed by the automaton after the first, meaning the timestamp of the second event is later than that of the first (assuming the input events are ordered chronologically). Besides automata, other approaches to CER include those that rely on (temporal) logic formalisms, trees or even hybrid systems [26].

A CER system typically runs multiple patterns simultaneously on a stream of input events and the stream of complex events might be composed of multiple event types and possibly overlapping events. As a result, the issue of optimizing the execution of all these patterns may become quite prominent. Similar to multi-query optimization in traditional database systems, CER patterns may share parts of their structure or of their computation. This is especially the case when a set of CER patterns is structured in the form of a hierarchy, where patterns defined at lower levels of the hierarchy may be used by patterns at higher levels. Obviously, in such cases, executing each pattern independently results in redundant computations, especially if the degree of "sharedness" is high, with one pattern being shared by multiple other patterns. Besides opening up an optimization potential, hierarchies are also used to express patterns in a concise manner and avoid cluttering. The interpretability dimension is another benefit of using hierarchies, by offering more compact pattern definitions.

Despite these benefits, the issue of hierarchies has not received much attention in the CER literature. The standard approach until now has been to employ a bottom-up, "event passing" mechanism [15, 16, 43, 53]. Patterns at lower levels of the hierarchy perform recognition and, upon detecting a complex event, they package this new event, adding a timestamp to it and push it upwards so that it becomes available for processing by patterns at higher levels. This approach is highly intuitive and easy to implement, thus gaining significant popularity. On the other hand, there are several issues with such a bottom-up approach. The number of events that need to be processed can multiply, especially if patterns at lower levels have high selectivity. More important, though, is the issue of assigning a "correct" timestamp to the intermediate, generated events. By assigning a single timepoint as a timestamp, the hierarchy becomes unsound, in the sense that it might detect complex events that do not exist. By using intervals for timestamps, the semantics of the hierarchy may be corrected, but up to a point. We need to make sacrifices as far as the expressive power of the patterns.

For these reasons, we explore a different "top-down" strategy. We first compile the patterns of a hierarchy into (symbolic) automata. We then let the automata decide on their own when to ask for information from the lower levels. Specifically, when an automaton reaches a point where information from another sub-automaton (corresponding to a sub-pattern) is required, the automaton freezes and delegates control to the sub-automaton. When the sub-automaton detects a complete match, control is transferred back to the original automaton. If multiple patterns share the same sub-automaton, then we also synchronize their execution so that the sub-automaton informs all of its "subscribers" for its status. By following such a top-down

approach, using run sharing and synchronization, we essentially force all automata to work on the “lossless” input stream directly, without intermediate, “lossy” events. We thus avoid the issue of unsound semantics.

We investigate the behavior of the Run Sharing and Synchronization (RSS) approach. We compare it against a naive, baseline method where patterns are treated independently, with the hierarchy essentially being flattened. Our contributions may be summarized as follows:

- We formalize the concept of hierarchies of patterns and automata.
- We introduce a new, top-down method, called Run Sharing and Synchronization (RSS), and show that it does not face any such limitations.
- We experimentally compare it against a naive implementation.

2.2.2 Background and problem formulation

In this section, we provide the necessary background for our work. First, we describe how patterns are defined. As our basis, we use symbolic patterns ($SyPa$), which are extensions of regular expressions, where transitions are equipped with symbolic predicates. We initially present the definition of symbolic patterns in the absence of hierarchies. We also present a computational model for symbolic patterns, called symbolic finite automata (SFA). We then show how $SyPa$ may be extended into hierarchical symbolic patterns ($SyP[H]a$), which constitute the building blocks of hierarchies of patterns. The distinctive feature of $SyP[H]a$ is that they can refer, in their definition, to other, previously defined $SyP[H]a$. We subsequently show how SFA can also be extended into hierarchical symbolic finite automata ($hSFA$) and how we can construct hierarchies of such automata. Again, the idea is that transitions in $hSFA$, besides being equipped with a guard in the form of a symbolic predicate, may also refer to other automata. We also prove the following:

- Every $SyP[H]a$ is equivalent to its “flattened” version, i.e., to a $SyPa$, where any sub-expressions have been replaced with their definitions.
- Every $hSFA$ is equivalent to its “flattened” version, i.e., to a SFA , where any sub-automata have been replaced with their definitions.
- For every $SyP[H]a$ there exists an equivalent $hSFA$, i.e., a hierarchy of patterns can be converted to an equivalent hierarchy of automata.

To facilitate the exposition of hierarchies (both of patterns and automata), we have intentionally omitted several constructs typically used in automata-based CER systems. For example, the predicates used in $SyPa$ and SFA are unary, in the sense that they are applied only to the current event being processed by them. On the other hand, it is often the case that CER patterns need to apply n -ary predicates to both the current event and past events. To enable this functionality, automata need to be equipped with some form of *memory*. CER automata can also act as transducers, in the sense that they can produce an *output* (i.e., a marking of all input events) to designate which input events belong to a detected match, as we may choose to ignore/skip some of them. *Negation* (i.e., the absence of events satisfying some predicate) is also a common operator. Finally, most automata work by assuming that the timestamps of the input events are single timepoints and that there exists a total order among them. In some applications, though, we need to be able to handle durative events, whose timestamps

typically assume the form of *intervals*. In this case, the input events might be overlapping, constituting a partial order.

Our basic framework will be one based on *SyPa*, i.e., patterns are defined as regular expressions whose terminal conditions are logical predicates applied on the current event each time. Note that *SyPa* make the following assumptions:

1. Input events are instantaneous.
2. At most one event can occur at every timepoint.
3. A complex event is assumed to have occurred at a single timepoint. Only the occurrence of a complex event is reported, not its constituent input events.
4. All patterns are unary, i.e., logical predicates are applied only on the current event, but not to past events. Patterns have no memory.

Ideally, we would like our base framework to be extended towards the following directions:

1. Add the capability of handling events with *intervals* ($SyP[I]a$).
2. Add the capability for patterns to mark certain events as belonging to a complex event by allowing a pattern to produce *output* ($SyP[O]a$).
3. Add the capability for patterns to be defined in a *hierarchical* manner ($SyP[H]a$).
4. Add *negation* as a pattern operator ($SyP[N]a$), while maintaining consistent semantics.
5. Add *memory* to patterns so that they can relate multiple events at different timepoints ($Sy[M]Pa$).

We will use the acronym *SyPa* to denote a simple symbolic pattern. When we add properties, we will use the notation $Sy[+X]P[+Y]a$, where X, Y are properties: I for interval, O for output, H for hierarchies, N for negation, M for memory. For example, symbolic patterns with intervals, output and negation are denoted as $SyP[ONI]a$. The final goal is to accommodate full $Sy[M]P[HONI]a$, i.e., **S**ymbolic **P**atterns with **M**emory, **H**ierarchies, **O**utput, **N**egation and **I**ntervals. We will start from the “simple” case of simple *SyPa* and show how we can move to $SyP[H]a$, i.e., patterns with hierarchies. The proofs are thus simplified substantially, due to lack of heavy notation. Then we will discuss to what extent our proposal can accommodate output, memory, negation and intervals.

We first describe *SyPa*. We assume that each terminal expression of a *SyPa* is a Boolean expression whose predicates are in the form of a relation P . We also assume that all possible input events constitute a universe $\mathcal{U}$. With respect to the temporal model, we assume a discrete model of timepoints. We also assume that each event (input or complex) has a timestamp and this timestamp is composed of a set of (at least one) timepoints. The minimum and maximum element of a timestamp constitute its interval.

Definition 1 (timestamp). A *timestamp* is a set of distinct, positive integers, $ts = \{t : t \in \mathbb{Z}^+\}$.

Definition 2 (event). An *event* is a tuple, consisting of an element u from a universe $\mathcal{U}$ and a timestamp, $e = (u, ts)$, where $u \in \mathcal{U}$ and ts a timestamp.

Typically, an event may have a timestamp with multiple timepoints, as in the definition above. We will be working mainly with instantaneous events, but we retain this definition, because it will be useful when we discuss the event passing algorithm.

We can define unary relations P on the elements of $\mathcal{U}$ and use these relations, or combinations of them via Boolean connectives, as terminal expressions within a $SyPa$. The argument of P refers to the most recently read element of a string. We call such terminal expressions “conditions”.

Definition 3 (Condition). *Let $\top$ be a unary relation for which it holds that $u \in \top$, $\forall u \in \mathcal{U}$, i.e., this relation holds for all elements of the universe $\mathcal{U}$. A condition is then defined by the following grammar:*

- $\top$ is a condition.
- P , where P is a unary relation, is a condition.
- $\neg\phi$ is a condition, if ϕ is a condition.
- $\phi_1 \wedge \phi_2$ is a condition if ϕ_1 and ϕ_2 are conditions.
- $\phi_1 \vee \phi_2$ is a condition if ϕ_1 and ϕ_2 are conditions. ◀

Definition 4 (Semantics of conditions). *Let $\mathcal{M}$ be a $\mathcal{V}$ -structure, $u \in \mathcal{U}$ an element of the universe of $\mathcal{M}$. We say that a condition ϕ is satisfied by u , denoted by $u \models \phi$, iff one of the following holds:*

- $\phi := \top$, i.e., $u \models \top$ for every element.
- $\phi := P$ and $u \in P$.
- $\phi := \neg\psi$ and $(u, v) \not\models \psi$.
- $\phi := \phi_1 \wedge \phi_2$, $(u, v) \models \phi_1$ and $(u, v) \models \phi_2$.
- $\phi := \phi_1 \vee \phi_2$, $(u, v) \models \phi_1$ or $(u, v) \models \phi_2$. ◀

$SyPa$ may now be defined as follows.

Definition 5 (Symbolic pattern ($SyPa$)). *A $SyPa$ over a $\mathcal{V}$ -structure $\mathcal{M}$ is inductively defined as follows:*

1. ϵ is a $SyPa$.
2. If ϕ is a condition (as in Definition 3), then ϕ is a $SyPa$.
3. If p_1 and p_2 are $SyPa$, then $p_1 + p_2$ is also a $SyPa$.
4. If p_1 and p_2 are $SyPa$, then $p_1 \cdot p_2$ is also a $SyPa$.
5. If p is a $SyPa$, then p^* is also a $SyPa$. ◀

Once we have a string S and a $SyPa$ p , the next step is to define the matches that p “induces” when applied on S . For the moment, we are going to define them simply as timestamps.

Definition 6 (candidate match). *A candidate match M is a set of timestamps $M = \{ts_1, \dots, ts_m\}$.*

We write $M = M_1 \cdot M_2$ for two matches M_1, M_2 to denote the fact that $M_1 \subset M, M_2 \subset M, M_1 \cap M_2 = \emptyset, M_1 \cup M_2 = M$ and $\max(M_1) < \min(M_2)$.

Definition 7 (match induction). *Let p be a $SyPa$, S a string and M a candidate match. We use $(p, S) \vdash M$ to denote a relation corresponding to the fact that the application of p on S induces M or that M is a match of p on S .*

We now define how matches are induced by patterns.

Definition 8 (Semantics of $SyPa$). *Let p, p_1, p_2 be $SyPa$ over a $\mathcal{V}$ -structure $\mathcal{M}$, ϕ a condition, S a string constructed from elements of the universe of $\mathcal{M}$, M a candidate match of p on S . We define the relation $(p, S) \vdash M$ as follows:*

1. $(\epsilon, S) \vdash M$ iff $S = \epsilon$ and $M = \emptyset$.
2. $(\phi, S) \vdash M$ iff $\phi \neq \epsilon, S = u, u \models \phi$ and $M = \{u.ts\}$.
3. $(p_1 \cdot p_2, S) \vdash M$ iff $S = S_1 \cdot S_2$ and $M = M_1 \cdot M_2 : (p_1, S_1) \vdash M_1$ and $(p_2, S_2) \vdash M_2$.
4. $(p_1 + e_2, S) \vdash M$ iff $(p_1, S) \vdash M$ or $(p_2, S) \vdash M$.
5. $(p^*, S) \vdash M$ iff

$$\begin{cases} S = \epsilon \text{ and } M = \emptyset \\ S = S_1 \cdot S_2, M = M_1 \cdot M_2 : (p, S_1) \vdash M_1 \text{ and } (p^*, S_2) \vdash M_2 \end{cases} \quad \text{or}$$

◀

Finally, the language accepted by a $SyPa$ and the matches detected by it are defined as follows.

Definition 9 (Language accepted and matches detected by a $SyPa$). *The language accepted by a $SyPa$ p is defined as $\mathcal{L}(p) = \{S \mid (p, S) \vdash M\}$ for some match M of p on the corresponding S . The set of all matches detected by a $SyPa$ p on a string S is defined as $\text{Match}(p, S) = \{M \mid (p, S) \vdash M\}$. ◀*

We now describe a computational model for $SyPa$ called symbolic finite automata (SFA). SFA are similar to classical automata, with their main feature being that their guards on the transitions are logical predicates instead of terminal symbols from an alphabet. Symbolic automata have been described and investigated elsewhere [3, 13]. We adjust them here to suit our purposes, but we do not provide details. Suffice it to say that they are closed under union, intersection, concatenation and Kleene-star. Symbolic automata with memory are not closed under negation, unless they are equipped with windows, i.e., a maximum time interval within which all input events of a match must occur.

Definition 10 (Symbolic Finite Automaton). *A symbolic finite automaton (SFA) over a $\mathcal{V}$ -structure $\mathcal{M}$ is a tuple (Q, q_s, Q_f, R, Δ) where*

- Q is a finite set of states,
- $q_s \in Q$ the start state,
- $Q_f \subseteq Q$ the set of final states and
- Δ the set of transitions.

A transition $\delta \in \Delta$ is a tuple (q, ϕ, q') , also written as $q, \phi \rightarrow q'$, where

- $q, q' \in Q$, where q is the source and q' the target state and
- ϕ is a condition, as per Definition 3 or $\phi = \epsilon$. ◀

We can describe formally the rules for the behavior of a *SFA* through the notion of configuration:

Definition 11 (Configuration of *SFA*). Assume a string $S = t_1, t_2, \dots, t_l$ and a *SFA* A consuming S . A configuration of A is a triple $c = [j, ts, q] \in \mathbb{N} \times \mathbb{T} \times Q$, where

- j is the index of the next event/character to be consumed,
- ts is the timestamp of t_j and
- q is the current state of A .

We say that $c' = [j', t', q']$ is a successor of c iff one of the following holds:

- $\exists \delta : \delta.source = q, \delta.target = q', \delta.\phi = \epsilon, j' = j, ts' = ts$, i.e., if this is an ϵ transition, we move to the target state without changing the index or the timestamp.
- $\exists \delta : \delta.source = q, \delta.target = q', t_j \models \delta.\phi, j' = j + 1, ts' = t_{j+1}.ts$, i.e., if the condition is satisfied according to the current event, we move to the target state, we increase the index by 1 and we update the timestamp. ◀

We denote a succession of configurations by $[j, ts, q] \rightarrow [j', ts', q']$, or $[j, ts, q] \xrightarrow{\delta} [j', ts', q']$ if we need to refer to the transition as well. For the initial configuration, before any elements have been consumed, we assume that $j = 1$ and $q = q_s$.

The actual behavior of a *SFA* upon reading a stream is captured by the notion of the run:

Definition 12 (Run of *SFA* over string/stream). A run ρ of a *SFA* A over a stream $S = t_1, \dots, t_n$ is a sequence of successor configurations $[1, t_1.ts, q_1] \xrightarrow{\delta_1} [2, t_2.ts, q_2] \xrightarrow{\delta_2} \dots \xrightarrow{\delta_n} [n+1, t_{n+1}.ts, q_{n+1}]$. A run is called accepting iff $q_{n+1} \in A.Q_f$. By $Match(\rho)$ we denote all the indices in the string, i.e., $Match(\rho) = \{i \in [1, n]\}$. ◀

The set of all runs over a stream S that A can follow is denoted by $Run(A, S)$ and the set of all accepting runs by $Run_f(A, S)$.

Finally, we can define the language of a *SFA* as the set of strings for which the *SFA* has an accepting run, starting from an empty configuration. Similarly, the matches of a *SFA* on a string are the matches the *SFA* “produces” by marking the input elements as it reads the string, starting from an empty configuration.

Definition 13 (Language recognized and matches detected by *SFA*). We say that a *SFA* A accepts a string S iff there exists an accepting run $\rho = [1, t_1.ts, q_1] \xrightarrow{\delta_1} [2, t_2.ts, q_2] \xrightarrow{\delta_2} \dots \xrightarrow{\delta_n} [n+1, t_{n+1}.ts, q_{n+1}]$ of A over S , where $q_1 = T.q_s$.

The set of all strings accepted by A is called the language recognized by A and is denoted by $Lang(A)$.

The set of matches detected by A on a string S is defined as $Match(A, S) = \{Match(\rho) \mid \rho \in Run_f(A, S)\}$. ◀

Theorem 1. *For every $SyPa$ p there exists an equivalent SFA A , i.e., a SFA such that $\mathcal{L}(p) = L(A)$ and $Match(p, S) = Match(A, S)$ for every string S .*

Based on $SyPa$, we can now define hierarchies of symbolic patterns. Hierarchies are composed of extensions of $SyPa$, called $SyP[H]a$. $SyP[H]a$ are essentially the same as $SyPa$ with one addition. A terminal expression can be ϵ or a condition ϕ , as in $SyPa$. It may also be an event type. Within a hierarchy, each pattern is assigned an event type, besides those of the input events. Inside a $SyP[H]a$, when a terminal expression is such an event type, this implies that the pattern corresponding to this event type should be used. Since $SyPa$ are by definition compositional, the addition of event types is essentially syntactic sugar and straightforward.

Definition 14 (Hierarchical symbolic pattern ($SyP[H]a$)). *Let $\mathcal{M}$ be a $\mathcal{V}$ -structure with universe $\mathcal{U}$. Let $\mathcal{E} = \{et_1, et_2, \dots, et_m\}$ be a finite set of event types. A $SyP[H]a$ over $\mathcal{M}$ and $\mathcal{E}$ is inductively defined as follows:*

1. ϵ is a $SyP[H]a$.
2. If ϕ is a condition (as in Definition 3), then ϕ is a $SyP[H]a$.
3. If $et \in \mathcal{E}$, then et is a $SyP[H]a$.
4. If p_1 and p_2 are $SyP[H]a$, then $p_1 + p_2$ is also a $SyP[H]a$.
5. If p_1 and p_2 are $SyP[H]a$, then $p_1 \cdot p_2$ is also a $SyP[H]a$.
6. If p is a $SyP[H]a$, then p^* is also a $SyP[H]a$. ◀

Definition 15 (Hierarchy of symbolic patterns). *Let $\mathcal{M}$ be a $\mathcal{V}$ -structure with universe $\mathcal{U}$. Let $\mathcal{E} = \{et_1, et_2, \dots, et_m\}$ be a finite set of event types. A hierarchy $\mathcal{H}_p$ of symbolic patterns is a set of m hierarchical symbolic patterns ($SyP[H]a$), exactly one for each $et \in \mathcal{E}$.*

$$\mathcal{H}_p = \begin{cases} et_1 \rightarrow hp_1 \\ et_2 \rightarrow hp_2 \\ \dots \\ et_m \rightarrow hp_m \end{cases}$$

where hp_i are $SyP[H]a$ over $\mathcal{M}$ and $\mathcal{E}$. Let $SUB(p)$ denote the set of event types referenced by a $SyP[H]a$ p , i.e.,

$$SUB(p) = \begin{cases} \emptyset & \text{if } p = \epsilon \wedge p = \phi \\ et_i \cup SUB(hp_i) & \text{if } p = et_i \\ SUB(p_1) \cup SUB(p_2) & \text{if } p = p_1 + p_2 \\ SUB(p_1) \cup SUB(p_2) & \text{if } p = p_1 \cdot p_2 \\ SUB(p_1) & \text{if } p = (p_1)^* \end{cases}$$

We say that $\mathcal{H}_p$ has a cycle iff $\exists i \in [1, m] : et_i \in SUB(hp_i)$. We require that $\mathcal{H}_p$ has no cycles. ◀

Definition 16 (Semantics of $SyP[H]a$). *Let hp be a $SyP[H]a$ of a hierarchy $\mathcal{H}_p$, ϕ a condition, S, S' strings constructed from elements of the universe of $\mathcal{M}$ and event types from $\mathcal{H}_p \cdot \mathcal{E}$, M a candidate match of hp on S . We define the relation $(hp, S) \vdash (M, S')$ as follows:*

1. $(\epsilon, S) \vdash (M, S')$ *iff*
 - $S = \epsilon$;
 - $M = \emptyset$;
 - $S' = S$.
2. $(\phi, S) \vdash (M, S')$ *iff*
 - $\phi \neq \epsilon$;
 - $S = u$;
 - $u \models \phi$;
 - $M = \{u.ts\}$;
 - $S' = S$.
3. $(et_i, S) \vdash (M, S')$ *iff*
 - $(hp_i, S) \vdash (M, S'')$;
 - $S' = \{(et_i, [\min(S), \max(S)])\}$.
4. $(p_1 \cdot p_2, S) \vdash (M, S')$ *iff*
 - $S = S_1 \cdot S_2$;
 - $M = M_1 \cdot M_2 : (p_1, S_1) \vdash (M_1, S'_1) \text{ and } (p_2, S_2) \vdash (M_2, S'_2)$;
 - $S' = S'_1 \cdot S'_2$.
5. $(p_1 + p_2, S) \vdash (M, S')$ *iff*
 - $(p_1, S) \vdash (M, S') \text{ or } (p_2, S) \vdash (M, S')$.
6. $(p^*, S) \vdash (M, S')$ *iff*

$$\left\{ \begin{array}{l} S = \epsilon \text{ and } M = \emptyset \text{ and } S' = S \\ S = S_1 \cdot S_2 \\ M = M_1 \cdot M_2 : \{(p, S_1) \vdash (M_1, S'_1) \text{ and } (p^*, S_2) \vdash (M_2, S'_2)\} \\ S' = S'_1 \cdot S'_2 \end{array} \right. \quad \begin{array}{l} \text{or} \\ \text{and} \\ \text{and} \end{array}$$



Definition 17 (Language accepted and matches detected by a $SyP[H]a$). *The language accepted by a $SyP[H]a$ hp is defined as $\mathcal{L}(hp) = \{S \mid (p, S) \vdash (M, S')\}$ for some match M of p on the corresponding S and some generated string S' .*

The set of all matches detected by a $SyP[H]a$ hp on a string S is defined as $Match(hp, S) = \{M \mid (p, S) \vdash (M, S')\}$.

The set of all strings generated by a $SyP[H]a$ hp on a string S is defined as $\mathcal{G}(hp, S) = \{S' \mid (p, S) \vdash (M, S')\}$ for some M . ◀

Obviously, in each $SyP[H]a$ we can repeatedly replace all event types with their original definitions in order to get a “flattened” version, i.e., a $SyP[H]a$ which would be reduced to $SyPa$, without any event types. The definition for flattened $SyP[H]a$ is as follows.

Definition 18 (Flattened $SyP[H]a$). *Let $\mathcal{H}_p$ be a hierarchy. The flattened version of a $SyP[H]a$ hp from $\mathcal{H}_p$ is a $SyPa$ obtained by repeatedly replacing each et_i in hp with its corresponding hp_i until no et_i remain. Is is denoted by $f(hp)$.*

$$f(hp) = \begin{cases} \epsilon & \text{if } hp = \epsilon \\ \phi & \text{if } hp = \phi \\ f(hp_i) & \text{if } hp = et_i \\ f(p_1) + f(p_2) & \text{if } hp = p_1 + p_2 \\ f(p_1) \cdot f(p_2) & \text{if } hp = p_1 \cdot p_2 \\ (f(p))^* & \text{if } hp = p^* \end{cases}$$

◀

This is always possible since the hierarchy has no cycles.

As expected, we can show that $SyP[H]a$ are equivalent to their flattened versions.

Proposition (Equivalence of $SyP[H]a$ and flattened version). *For every $SyP[H]a$ hp , $\mathcal{L}(hp) = \mathcal{L}(f(hp))$. Also, $Match(hp, S) = Match(f(hp), S)$ for every S .*

We can now define a hierarchical version of SFA , called $hSFA$. We follow a similar approach. $hSFA$ are identical to SFA , with one addition. Within a hierarchy, all automata are assigned a unique event type. A guard on transition may simply be another event type. The idea is that, when a run of $hSFA$ encounters a transition with an event type, then it must enlist the automaton corresponding to that event type in order to resume its execution.

Definition 19 (Hierarchical Symbolic Finite Automaton). *A hierarchical symbolic finite automaton ($hSFA$) over a $\mathcal{V}$ -structure $\mathcal{M}$ and a finite set of event types $\mathcal{E}$ is a tuple (Q, q_s, Q_f, R, Δ) where*

- Q is a finite set of states,
- $q_s \in Q$ the start state,
- $Q_f \subseteq Q$ the set of final states and
- Δ the set of transitions.

A transition $\delta \in \Delta$ is a tuple (q, g, q') , also written as $q, g \rightarrow q'$, where

- $q, q' \in Q$, where q is the source and q' the target state and
- $g = \epsilon$ or $g = \phi$, where ϕ is a condition, as per Definition 3 or $g \in \mathcal{E}$. ◀

Definition 20 (Hierarchy of symbolic automata). *Let $\mathcal{H}_p$ be a hierarchy of symbolic patterns.*

A hierarchy $\mathcal{H}_a$ of symbolic automata is a set of $\mathcal{H}_p.m$ hSFA, exactly one for each $et \in \mathcal{H}_p.\mathcal{E}$.

$$\mathcal{H}_p = \begin{cases} et_1 \rightarrow ha_1 \\ et_2 \rightarrow ha_2 \\ \dots \\ et_m \rightarrow ha_m \end{cases}$$

where ha_i are hSFA over $\mathcal{H}_p.\mathcal{M}$ and $\mathcal{H}_p.\mathcal{E}$.

Definition 21 (Configuration of hSFA). Assume a string $S = t_1, t_2, \dots, t_l$ and a hSFA A of a hierarchy $\mathcal{H}_a$ consuming S . A configuration of A is a triple $c = [j, ts, q] \in \mathbb{N} \times \mathbb{T} \times Q$, where

- j is the index of the next event/character to be consumed,
- ts is the timestamp of t_j and
- q is the current state of A .

We say that $c' = [j', t', q']$ is a successor of c iff one of the following holds:

- there exists $\delta \in A.\Delta$ such that
 - $\delta.source = q$;
 - $\delta.target = q'$;
 - $\delta.g = \epsilon$;
 - $j' = j$;
 - $ts' = ts$;
 - i.e., if this is an ϵ transition, we move to the target state without changing the index or the timestamp.
- there exists $\delta \in A.\Delta$ such that
 - $\delta.source = q$;
 - $\delta.target = q'$;
 - $\delta.g$ is a condition ϕ ;
 - $t_j \models \phi$;
 - $j' = j + 1$;
 - $ts' = t_{j+1}.ts$;
 - i.e., if the condition is satisfied according to the current event, we move to the target state, we increase the index by 1 and we update the timestamp.
- there exists $\delta \in A.\Delta$ such that
 - $\delta.source = q$;
 - $\delta.target = q'$;

- $\delta.g = et_i \in \mathcal{H}_a.\mathcal{E}$
- $j' = j + k + 1$;
- $ts' = t_{j+k+1}.ts$;
- *there exists a sequence of successor configurations of automaton ha_i of $\mathcal{H}_a$*

$$[j, t_j.ts, q_1] \rightarrow \dots \rightarrow [j + k + 1, t_{j+k+1}.ts, q_{k+1}]$$

where $q_i \in ha_i.Q$, $q_1 = ha_i.q_s$, $q_{k+1} \in ha_i.Q_f$;

- *i.e., from $j + 1$ up to $j + k - 1$ the automaton corresponding to et_i begins from its start state and must finish in one of its final states.*

◀

Definition 22 (Run of $hSFA$ over string). A run ϱ of a $hSFA$ A over a string $S = t_1, \dots, t_n$ is a sequence of successor configurations $[1, t_1.ts, q_1] \xrightarrow{\delta_1} [j_2, t_{j_2}.ts, q_2] \xrightarrow{\delta_2} \dots \xrightarrow{\delta_{n'}} [n + 1, t_{n+1}.ts, q_{n'}]$. A run is called accepting iff $q_{n+1} \in A.Q_f$.

By $Event(\delta_i)$ we denote the event that triggered transition δ_i .

$$Event(\delta_i) = \begin{cases} \epsilon & \text{if } \delta_i.g = \epsilon \\ t_i & \text{if } \delta_i.g = \phi \\ (et, [t_{j_i}.ts, t_{j_{i+1}-1}.ts], [j_i, j_{i+1} - 1]) & \text{if } \delta_i.g = et \end{cases}$$

By $DS(\varrho)$ we denote the string derived from ϱ : $DS(\varrho) = Event(\delta_1), Event(\delta_2), \dots, Event(\delta_{n'})$.

By $index(e)$ we denote the indices of the event e , i.e., $index(e) = \{e.istart, e.iend\}$.

By $Match(\varrho)$ we denote all the indices in the string, i.e., $Match(\varrho) = \{i \in [1, n]\}$.

By $EMatch(\varrho)$ we denote all the indices in the derived string, i.e., $EMatch(\varrho) = \{in \mid in \in index(Event(\delta_i))\}$. ◀

Definition 23 (Language recognized and matches detected by $hSFA$). We say that a $hSFA$ A accepts a string S iff there exists an accepting run $\varrho = [1, t_1.ts, q_1] \xrightarrow{\delta_1} \dots \xrightarrow{\delta_{n'}} [n + 1, t_{n+1}.ts, q_{n'}]$ of A over S , where $q_1 = A.q_s$.

The set of all strings accepted by A is called the language recognized by A and is denoted by $\mathcal{L}(A)$.

The set of strings generated by a $hSFA$ A on a string S is defined as $\mathcal{G}(A, S) = \{DS(\varrho) \mid \varrho \in Run_f(A, S)\}$.

The set of matches detected by A on a string S is defined as $Match(A, S) = \{Match(\varrho) \mid \varrho \in Run_f(A, S)\}$.

◀

As with $SyP[H]a$, in each $hSFA$ we can repeatedly replace all event types with their original definitions in order to get a “flattened” version, i.e., a $hSFA$ which would be reduced to SFA , without any event types. The definition for flattened $hSFA$ is as follows.

Definition 24 (Flattened $hSFA$). *Let $\mathcal{H}_a$ be a hierarchy. The flattened version of a $hSFA$ A from $\mathcal{H}_a$ is a SFA obtained by repeatedly replacing each et_i in A with its corresponding ha_i until no et_i remain. Is denoted by $f(A)$. Replacement of a et_i transition means the following:*

- *The transition is removed.*
- *The automaton ha_i is inserted with all of its states and transitions.*
- *The final states of ha_i are flipped to normal states.*
- *The source of the removed et_i transition is connected to the start state of ha_i via an ϵ transition.*
- *The final states of ha_i are connected to the target of the removed et_i transition via ϵ transitions.*

◀

This is always possible since the hierarchy has no cycles.

As expected, we can show that $hSFA$ are equivalent to their flattened versions.

Proposition (Equivalence of $hSFA$ and flattened version). *For every $hSFA$ A , $\mathcal{L}(A) = \mathcal{L}(f(A))$. Also, $Match(A, S) = Match(f(A), S)$ for every S .*

Finally, we can show that a hierarchy of $SyP[H]_a$ can be compiled into an equivalent hierarchy of $hSFA$.

Theorem 2 (Equivalence of $SyP[H]_a$ and $hSFA$). *Let $\mathcal{H}_p$ be a hierarchy of $SyP[H]_a$. Then there exists a hierarchy of $hSFA$ $\mathcal{H}_a$ such that for every $SyP[H]_a$ of $\mathcal{H}_p$, hp_i , there exists a $hSFA$ of $\mathcal{H}_a$, ha_i , with*

- $\mathcal{L}(hp_i) = \mathcal{L}(ha_i)$.
- $Match(hp_i, S) = Match(ha_i, S)$ for every string S .
- $\mathcal{G}(hp_i, S) = \mathcal{G}(ha_i, S)$ for every string S .

2.2.3 Run sharing and synchronization

As already discussed, there are multiple approaches to implementing a hierarchy of $hSFA$. The first one would be to directly implement the notions of configurations and runs. The second is through an event passing mechanism. The third is with run sharing and synchronization. The first method would be highly inefficient, whereas the second one has several soundness issues.

One solution to the problem of soundness would be to work with “complete histories”, i.e., not just intervals, but intervals where we know, for each timestamp, the event it corresponds to. In order to do so, we would need to carry all the information about the Bs, not just their timestamps, which would be highly inefficient.

In fact, in [86] it has been shown that the only interval model that can avoid such soundness issues is one where complete histories are available. This work implicitly assumes that

patterns use the skip-till-next-match selection strategy. It is interesting to note that skip-till-next-match essentially implies negation. First, note that a pattern

$$A_1 \cdot A_2, \dots, A_n$$

when using skip-till-next-match, can actually be rewritten as

$$A_1 \cdot (!(\top^* \cdot A_2 \cdot \top^*))^* \cdot A_2 \dots (!(\top^* \cdot A_n \cdot \top^*))^* \cdot A_n$$

where $!$ stands for negation. On the other hand, we have shown that patterns with the standard regular operators and skip-till-any-match encounter no such problems. Therefore, it seems reasonable that negation (with its implied universal quantification) is at the heart of the problem.

A possible solution would be to attempt to compress complete histories and let automata work on them, thereby achieving soundness and restricting the performance penalty from complete histories. As discussed in the next section, we follow a different approach. Instead of creating intermediate, “lossy” events and push them upwards, we let automata dive directly into the original stream. Computation sharing is then achieved through run sharing and synchronization (RSS). With such an approach, we bypass completely any issues of soundness.

For the sake of brevity, we avoid showing the main RSS algorithm in its full details. We only provide a high-level overview. Each automaton in the hierarchy has some default runs. The default runs are those runs which, before we start processing a new input event, are always in some start state. These are essentially the runs that are ready to initiate a new candidate match, since we require our patterns to be able to start a match at any point in the stream. Each configuration of an automaton ha_i is now a 5-tuple, composed of the following:

1. The index of the next event to be processed by ha_i ;
2. The timestamp of the next event to be processed by ha_i ;
3. The state that the automaton ha_i is in;
4. A set of runs of any sub-automata;
5. A set of runs of any super-automata, i.e., automata that track ha_i , which means that ha_i is a sub-run of them.

As long as ha_i follows normal transitions, the set of sub-runs is empty. When it encounters a et_j transition, it “drops an anchor” to the default runs of ha_j . In other words, the default runs of ha_j become sub-runs of the current run of ha_i . This is similar to a fork in operating systems terminology. At the same time, the current run of ha_i becomes a tracker of these default runs. From that point on, the current run of ha_i is frozen and any new events are processed by its sub-runs. This process repeats iteratively if one of these sub-runs also needs to drop an anchor to yet other runs at lower levels. If one of these sub-runs ever reaches a final state, then the current run of ha_i thaws and is allowed to resume its execution. Note that if multiple automata, say both ha_i and $ha_{i'}$ need to freeze and drop an anchor to ha_j at the same time, they can do so. The default runs of ha_j will be tracked by both ha_i and $ha_{i'}$. Thus, ha_i and $ha_{i'}$ are synchronized and do not need to create separate sub-runs. Only one set of sub-runs of ha_j is created and shared among them.

Theorem 3 (Correctness of the RSS Algorithm). *Assume a string S and a hSFA ha_i of a hierarchy $\mathcal{H}_a$ consuming S . The RSS algorithm reports a match M for S on ha_i iff $S \in \mathcal{L}(ha_i)$ and $M \in Match(ha_i, S)$.*

2.2.4 Empirical evaluation

In this section, we present some preliminary results on the RSS approach. We investigate the performance benefits of RSS compared to a naive system which evaluates all patterns independently, as if they were flattened. In order to work in a controlled setting, we experiment with synthetic datasets and patterns. Our datasets are composed of simple input events with event types A , B , etc.

Our patterns are mostly sequential. For example, we use the following pattern at level $l = 1$:

$$p_1 := B \cdot C \cdot D \cdot E \quad (1)$$

Then, we create 10 patterns at level $l = 2$ of the following form:

$$q_i := A \cdot p_1 \cdot F$$

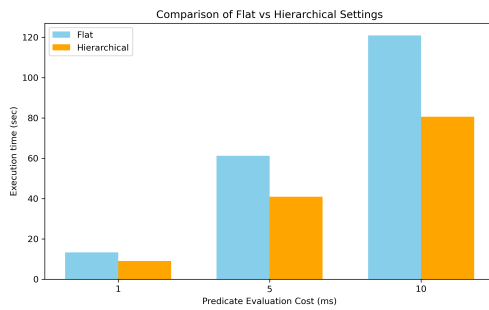
with $i \in [1, 10]$.

We evaluate this hierarchy first by flattening all q_i . Then we evaluate it by applying our RSS algorithm. We measure the throughput of the system in each case. Besides throughput, we also measure other metrics, such as the number of runs/clones created and the number of predicate evaluations. We expect that a hierarchical approach would reduce the number of runs and predicate evaluations which constitute the bulk of the total cost of processing a stream. We also tweak the cost of each predicate evaluation. We artificially set the cost to various values in order to gauge the impact of that cost on the total system performance. Our expectation is that the effect of an optimization algorithm, like RSS, would be more pronounced in cases where predicates are expensive.

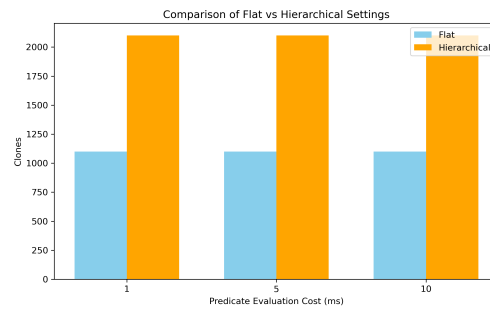
Figure 80 presents the relative results. We see that in every case, the performance of the hierarchical approach is $\approx 30\%$ better than that of the flat (Figure 11a). This is due to the decreased number of predicate evaluations, as shown in Figure 11c. The synchronization overhead of the RSS algorithm is shown in Figure 11b, where we see that it creates substantially more clones. However, this is offset by the decreased number of predicate evaluations. As expected, the number of clones and predicate evaluations is not affected by the cost of predicate evaluation.

2.3 Sequencing in the Run-Time Event Calculus

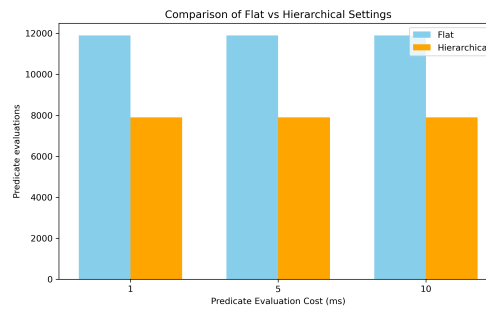
We have been developing a comprehensive framework for complex event recognition (CER) over high-velocity data streams. The basis of this framework is the Run-Time Event Calculus, a logic programming implementation of the Event Calculus that has been optimised for stream reasoning [50]. Streams often include events with delayed effects. Consider CER for maritime situational awareness, where a trawling activity is said to end some minutes after a ‘change in heading’ event, provided that no other event of this type has taken place in the meantime. We proposed a formal computational framework that handles streams including events with delayed effects [51]. We presented the syntax, semantics and reasoning



(a) Execution time as a function of predicate evaluation cost.



(b) Clones as a function of predicate evaluation cost.



(c) Predicate evaluations as a function of predicate evaluation cost.

Figure 11: Experimental results.

algorithms of our proposed framework, and demonstrated its correctness and complexity. Furthermore, we presented a reproducible analysis on large synthetic and real data streams, concerning CER on maritime situational awareness, and compared the efficiency of our approach with state-of-the-art systems. Our results demonstrated that our framework is capable of reasoning over very large streams, including events with delayed effects, while outperforming the state-of-the-art, often by orders of magnitude.

A fundamental operator for CER is “sequencing”, expressing that two activities take place one after the other. There is no consensus on a universal definition for sequencing. We provided a set of required properties for a sequencing operator for CER, i.e., an interval-based semantics, required for durative activities, and associativity, required to express activity hierarchies [48]. Furthermore, we proposed a sequencing operator that satisfies all requirements, as opposed to the ones in the literature, and we implemented our operator in RTEC. We compared our operator both theoretically and empirically with state-of-the-art approaches, demonstrating its benefits and limitations.

Complex event definition construction is a highly involved task, as these definitions often involve numerous spatio-temporal constraints, while labels that may be used for learning them automatically are hard to obtain. To address this issue, we proposed a method that generates complex event definitions from natural language descriptions using pre-trained Large Language Models (LLMs) [38]. In order to assess the quality of LLM-generated definitions, we proposed a novel similarity metric, which reflects the human effort required to correct them. We presented a thorough experimental evaluation of our approach on the maritime domain, demonstrating its effectiveness.

A program for CER, constructed with or without the use of LLMs, may lead to suboptimal reasoning, as the program developer cannot be expected to make perfect use of the language of RTEC. To address this issue, we developed a compiler for RTEC that optimises a given temporal specification expressing a CER program [49, 47]. The compiler identifies, in a process transparent to the user, the so-called “simple fluent-value pairs” that may be translated to equivalent “statically” determined fluent-value pair representations. The motivation stems from the fact that the latter representations lead to more efficient reasoning than the former. Our reproducible empirical analysis showed that the compilation process reduces the CER programs found in the literature by orders of magnitude, leading to significant reasoning efficiency gains.

2.4 Components for Critical Maritime Event Forecasting and Resolution

Two distinct components for forecasting and resolving critical maritime events are being developed by Kpler within the Maritime Use Case:

1. **MAR_1: Collision forecasting and rerouting**
2. **MAR_2: Hazardous weather routing**

Development of *MAR_1* was completed by M 18 and fully documented in Deliverable D4.1. Consequently, the present section of Deliverable D4.2 concentrates on advances in *MAR_2*—the hazardous weather routing component—building upon the progress outlined

in D4.1. Therefore the following subsections will focus on the presentation of the fusion of weather conditions with historical vessel traffic patterns, the final routing algorithm and evaluation experiments conducted to measure performance and efficiency of the proposed weather routing approach.

2.4.1 MAR_2: Hazardous weather routing

The hazardous weather routing component leverages collective vessel–fleet intelligence by combining historical and forecasted meteorological data with historical mobility patterns to address routing challenges posed by severe maritime weather events. Within the framework of the Maritime Use Case, Kpler is developing this capability for both manned and unmanned vessels. A robust hazardous weather routing solution is of substantial importance for international shipping, as it enhances navigational safety and reduces route disruptions that may have cascading impacts on the global supply chain. The increasing frequency and severity of extreme weather events attributed to climate change heighten the risks to maritime operations. Since maritime trade routes underpin the transport of vital commodities such as crude oil, liquefied natural gas (LNG) and other essential cargoes, ensuring that vessels can dynamically adjust their routes to avoid hazardous conditions is critical for maintaining the resilience and efficiency of global maritime logistics.

This section summarizes the methodology for integrating historical vessel traffic patterns with forecasted weather conditions when computing safe and efficient maritime routes. The early processing steps—solution–space generation and ingestion of meteorological forecasts—have been reported in detail in Deliverable D4.1 and are recapitulated here only briefly. A discretised representation of the navigable sea area is created using the Uber H3 hierarchical hexagonal grid and masking out land cells. Meteorological data from the Copernicus and NOAA services are ingested through automated pipelines and interpolated onto this grid together with historical AIS data. Subsequently, based on the fused weather and historical vessel transitions, the mobility graphs are extracted. In a final step a cost function for the A* routing algorithm is defined to route vessels towards their destination based on forecasted weather conditions and historical vessel traffic patterns. The weather routing service outputs may be disseminated through Application Programming Interfaces (APIs) to Maritime Situational Awareness Platforms [27], vessels sailing around the globe and critical situation management platforms (CREXDATA). Figure 12 presents an overview of the proposed methodology deployed to solve the weather routing problem.

Fusion of weather conditions with historical vessel traffic patterns We harvest collective fleet intelligence from historical AIS data in the MarineTraffic databases that are fused with historical weather data from the NOAA and Copernicus services. The Patterns of Life (PoL) framework [72] is extended to assign each AIS message to a weather category based on the Douglas Sea and Beaufort scales. Traffic statistics such as density, average speed, heading and predominant destinations are then computed per H3 cell and per weather category, yielding weather–aware PoL datasets. Discretising weather features into categorical states (Table 4) ensures statistical robustness when combining sparse AIS data with the weather data.

Figure 13 shows an example of how traffic patterns vary across wave–height classes in the Gulf of Mexico.

Table 4: Scale categories and statistical analysis of weather data from September 2022 in the North Atlantic.

Feature	Units	Scale Categories	Area Statistics
Wind speed	knots	0-16: Calm/ Breeze 17-40: Gale 40+: Storms	min: 0.000
			25%: 6.210
			median: 10.307
			75%: 14.996
			max: 89.900
			sample: 4,705,920
Wave height	m	0-1.25: Low 1.25-6: Moderate 6+: High	min: 0.000
			25%: 1.230
			median: 1.700
			75%: 2.290
			max: 19.100
			sample: 123,561,720

Vessel traffic Multigraph Generation The weather-aware PoL statistics are used to build a *MultiGraph* over the H3 grid. Four directed graphs share the same node topology but differ in edge weights derived from the frequency and distribution of observed vessel transitions under overall traffic (no weather), low-, moderate- and high-severity weather conditions. Transition weights $w_T(n_i, n_j)$ are estimated from AIS counts via Eq. (2). A one-step neighbour connectivity rule and land mask ensure that the graph remains connected even where AIS data are sparse. Figure 14 illustrates an example of the resulting topology for the Florida Peninsula.

$$w_T(n_i, n_j) = \begin{cases} 1 - \frac{t(n_i, n_j)}{\sum_{k=1}^{E_{n_i}} t(n_i, n_k)} & n_i \rightarrow n_j \neq 0 \\ 1 & n_i \rightarrow n_j = 0 \end{cases} \quad (2)$$

where:

$t(n_i, n_j)$ denotes the number of vessel traffic transitions for the directed edge e_{n_i, n_j} from node n_i to node n_j ;

E_{n_i} is the total number of outgoing edges of node n_i ;

$w_T(n_i, n_j)$ is the transition weight based from node n_i to node n_j based on historical vessel traffic data.

Weather-informed vessel traffic routing Routing is performed using a modified A* algorithm that incorporates dynamic weather forecasts and historical traffic patterns. The evaluation function adds traffic and weather penalties weighted by a user-tunable factor F_T (Eq. (3)).

$$f_{WT}(n_j, t) = g(n_i) + F_T \cdot w_T(n_i, n_j) + (1 - F_T) \cdot w_W(n_i, n_j, t) + h(n_j, n_D) \quad (3)$$

where:

F_T is the factor determining the importance of historical traffic patterns against weather penalties, $F_T \in [0, 1]$;

$g(n_i)$ is the cumulative cost up to current position n_i ;

$w_T(n_i, n_j)$ is the traffic weight based on historical traffic patterns defined in (2) and is independent of time $w_T(n_i, n_j) \in [0, 1]$;

$w_W(n_i, n_j, t)$ is the weather penalty at the destination node n_j at the expected arrival timestamp t , given the vessel defined travel speed v and the edge length $\tau(n_i, n_j)$; The weather-based weights have a temporal forecast horizon of 10 days in the future, according to the available capabilities of the Copernicus [20], [21] and NOAA [23] services. If the vessel is expected to arrive after that threshold, traffic-only weights are used instead;

$h(n_j, n_D)$ is a user defined heuristic function. Here we use the minimum grid distance between the next node and the destination node $n_j \rightarrow n_D$ as it is derived by the H3 grid traversal function [76].

$$w_W(n_i, n_j, t) = F_{WT} \cdot w_{WT}(n_i, n_j, c(t)) + (1 - F_{WT}) \cdot w_{W_{scaled}}(n_i, t) \quad (4)$$

F_{WT} is the user-defined factor for weighing weather-informed vessel transitions and weather forecast features, in general $F_{WT} \in [0, 1]$ (in the context of this study we use $F_{WT} = 0.5$);

$w_{WT}(n_i, n_j, c(t))$ is the weight of historical vessel transitions between nodes n_i and n_j for a given weather category $c(\cdot)$ at the forecasted weather conditions at time t with $w_{WT}(n_i, n_j, c(t)) \in [0, 1]$;

$w_{W_{scaled}}(n_i, t)$ is the cumulative penalty based on the scaled weather forecast features at current node n_i and time t according to (5) and (6) with $w_{W_{scaled}}(n_i, t) \in [0, 1]$.

$$s_f(k, t) = \frac{f(k, t) - f_{\min}(k)}{f_{\max}(k) - f_{\min}(k)} \quad (5)$$

where:

$f(k, t)$ is the forecast value for weather feature k at time t ;

$f_{\min}(k)$ is the minimum value for weather feature k in historical data;

$f_{\max}(k)$ is the maximum value for weather feature k in historical data.

Weather penalties combine historical transition costs and scaled forecast features (Eqs. (4)–(6)), while a heuristic based on H3 grid distance guides the search.

$$w_{W_{scaled}}(n_i, t) = \sum_{k=1}^K f_{w(n_i)}(k) \cdot s_f(k, t) \quad (6)$$

where:

$w_{W_{scaled}}(n_i)$ is the cumulative weather forecast feature weight in node n_i for all considered sea weather features [wind, wave height, currents] with $w_{W_{scaled}}(n_i) \in [0, 1]$;

$f_{w(n_i)}(k)$ is the user defined contribution of each weather feature $k \in K$ with $f_{w(n_i)}(k) \in [0, 1]$;

$s_f(k, t)$ is the scaled weather feature forecast by equation (5).

Vessel navigable route generation and deployment The path returned by the A* search is simplified with the Douglas–Peucker algorithm to produce a navigable sequence of waypoints. Each waypoint is annotated with the vessel’s speed, course and estimated time of arrival. The weather routing service is deployed on Amazon Web Services, ingesting 10-day forecasts and falling back to traffic-only weights when forecasts are unavailable. An API accepts queries specifying origin or destination (port or point), vessel speed and departure time and returns multiple routing alternatives. Figure 15 shows an example where the Weather–Traffic–Optimised Route (green polyline) is contrasted with the baseline shortest route (orange dashed).

When the forecast horizon is exceeded, only the traffic term in Eq. (3) is applied. Cumulative cost and arrival times are updated as in Eqs. (7)–(8). When forecasted conditions are severe, the algorithm favours paths that minimise exposure, balancing safety against route length.

$$g(n_k) = g(n_i) + F_T \cdot w_T(n_i, n_k) + (1 - F_T) \cdot w_W(n_i, n_k, t(n_i)) \quad (7)$$

$$t(n_j) = t(n_i) + \Delta t(n_i, n_j) \quad (8)$$

Evaluation The proposed approach was evaluated using 40 origin–destination pairs in the Gulf of Mexico and U.S. East Coast during September 2022, a month characterised by frequent Atlantic hurricanes. Weather routing increased the average route length by 3.26 % compared with the shortest path but reduced very extreme weather penalties by 35 % and extreme penalties by 15% (Table 5), indicating substantially improved safety for modest additional distance.

A scalability test with 1,200 random queries across the North Atlantic showed that response latencies scaled linearly with route length and remained below 4 seconds even for journeys up to 6000km (Fig. 16). These results demonstrate that the method delivers robust weather-aware routing with acceptable computational overhead.

Table 5: Results for the hazardous weather routing solution against the shortest path method.

Routing	Route Length (km) Average	Weather Penalties Median	Maximum Weather Penalty Median	Very Extreme Weather Penalties (>0.5)	Extreme Weather Penalties (>0.33)
Shortest (S)	2236	0.200	0.465	0.260	0.271
Weather (W)	2309	0.166	0.359	0.169	0.230
W / S (%)	3.26%	-16.97%	-22.70%	-35.03%	-15.21%

2.5 Discussion & Lessons Learnt

We presented a method for multi-resolution Complex Event Forecasting based on forward recognition. This method of forecasting via forward recognition offers several advantages:

- We can run various time series forecasting methods, depending on the dataset and the domain. In some cases, a simple ARIMA might be enough, whereas in others we might need a time series foundation model.
- We can perform time series aggregation and/or smoothing so that we can work at various resolutions.
- We can run forward recognition for multiple values of window center and size.
- For each method, we can estimate the Volume under Curve (VUC) as a forecasting metric.

Our work presented in this section underlines the fact that Complex Event Forecasting is still a relatively unexplored area, despite the fact that time series forecasting is well-established. Complex Event Forecasting can borrow several ideas from time series forecasting, but these ideas cannot be simply copied. There is a need for extensive experimentation, both theoretical and practical.

We have also presented a system for hierarchical CER. Contrary to previous approaches that relied on a bottom-up process of generating intermediate events and pushing them from the lower to the higher levels of a hierarchy, our method follows a top-down logic, whereby automata that need other sub-automata freeze their execution and transfer control to them, when required. This approach of run sharing and synchronization allows us to overcome several limitations on the expressive power of patterns used in hierarchies. At the same time, we reap the benefits of computation sharing.

Our presentation focused on $SyP[H]_a$, which are relatively simple patterns, without output, memory, negation or intervals. The questions that arise at this point concern the extent to which our approach can incorporate these extra constructs. As far as the output is concerned, the answer is affirmative. Adding the ability to mark certain events as relevant or irrelevant affects only the way matches are produced. Since, in our framework, matches do not affect

in any way the semantics of patterns, the addition of output on the conditions of patterns and on the guards of the automata, would not affect it any way. Therefore, $SyP[HO]_a$ are readily accommodated by our framework. The same is true for the functionality of memory. Symbolic transducers with memory have already been explored elsewhere [3]. They have been shown to be closed under union, concatenation and Kleene-star, but not under negation, unless a window is used. Using such automata does not alter our framework in any way. Thus, $Sy[M]P[HO]_a$ could also be handled by our system. With respect to negation, it could be fully supported if patterns and automata without memory are used. As note previously, though, if memory is required, then negation could only be used with a window that restricts the slice of the stream where a match is to be detected. As a result, $Sy[M]P[HON]_a$ can be accommodated, with the restriction that a window is necessary if memory is also required.

Finally, with respect to full $Sy[M]P[HONI]_a$, it remains an open issue whether our framework can handle them. The temporal model would need to change significantly in order to be able to represent durative events, with some of them possibly being contemporaneous or overlapping. We have, however, reasons to believe that durative events can be handled in an efficient manner, provided that we know each start timepoint of a durative event as soon as it “occurs”. In other words, we do not have to wait for the event to complete in order to be informed about its start timepoint. This is future work.

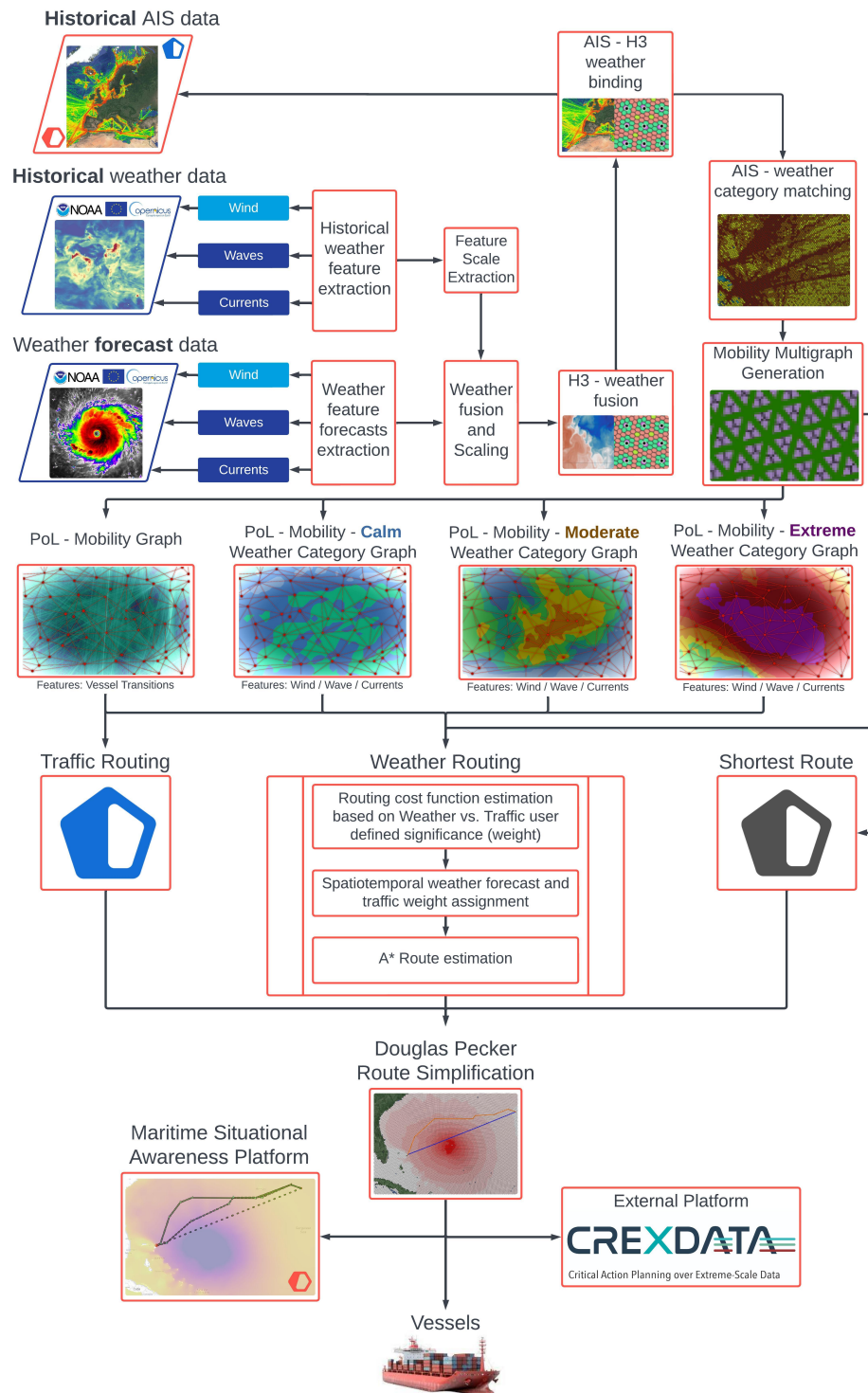


Figure 12: Weather routing architecture overview

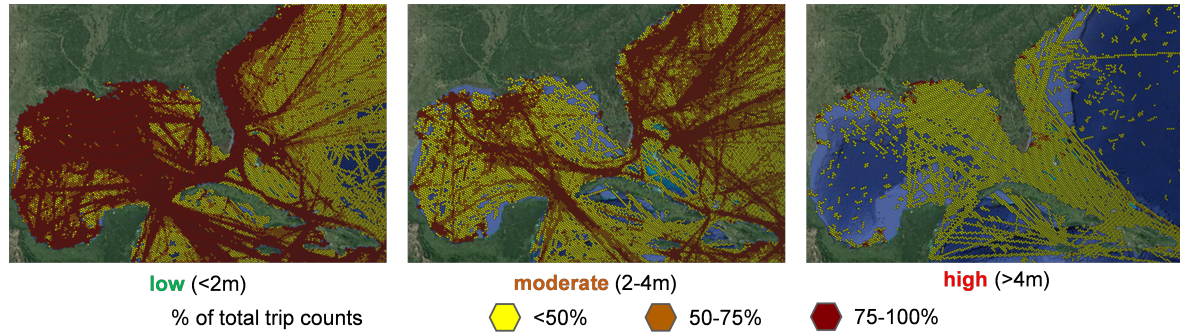


Figure 13: Example vessel traffic patterns based on historical wave heights (low: <2m, moderate: 2-4m, high: >4m).

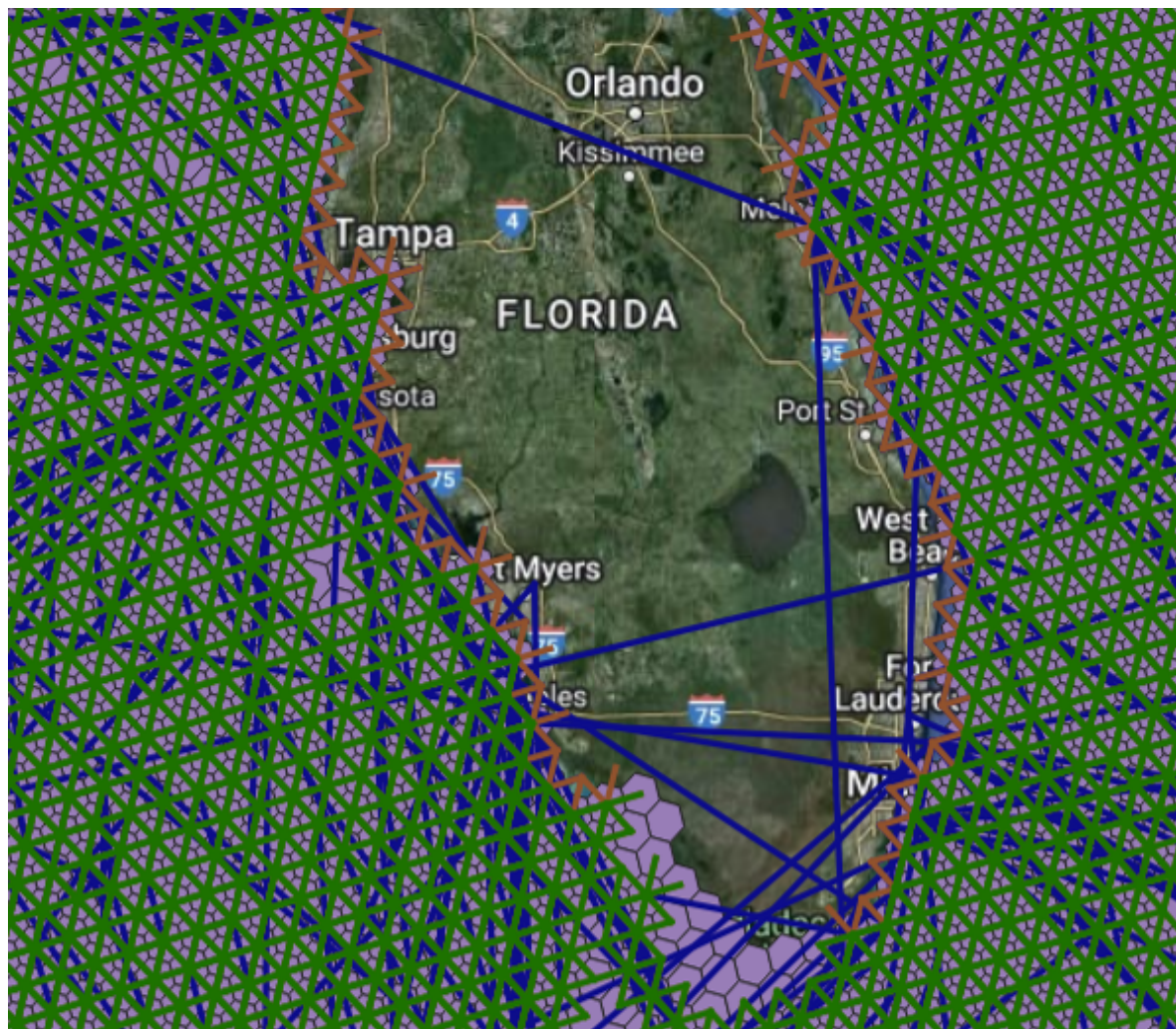


Figure 14: Routing graph around the Florida peninsula. Purple: H3 nodes; green: valid 1-step edges; blue: discarded multi-step transitions; brown: masked edges excluded by sea mask.



Figure 15: Example of the weather routing application output on the user interface.

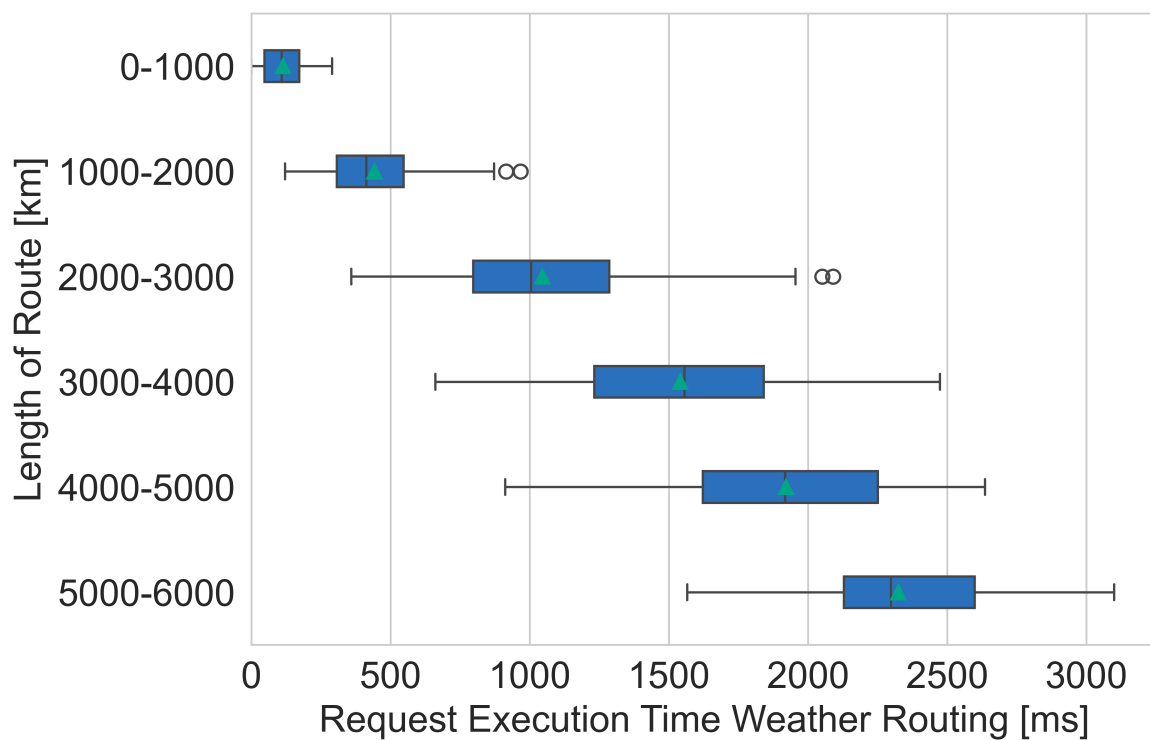


Figure 16: Weather routing request latency [ms] as a function route length [km].

3 Interactive Learning for Simulation Exploration

As presented in the DoA, the targets of task “T4.2 Interactive learning for simulations exploration” were to:

- Develop algorithms for guiding large-scale simulations towards desired ends.
- Simulate course corrections and adaptations to a changing environment.
- Avoid exhaustive search of the solution space by incorporating faster and effective solutions.
- Promote interactive learning to allow users to be more active in guiding the simulations.

During the first half of the project, we explored the relevant interactive learning scenarios for each use-case of CREXDATA. The deliverable “D4.1 Initial Report on Complex Event Forecasting, Learning and Analytics” described the simulator parameters that are subject to exploration, either for calibration using real-world data, or to optimize and search the simulations’ space. For each case, we outlined appropriate algorithm categories, i.e. Informed Search, Early Time Series Classification, and Reinforcement Learning.

During the second half of the project, we implemented the specified algorithms, and integrated them as AI Studio operators, allowing their utilization in the workflows of each use-case. In the following subsections, we describe the work performed for each use-case in detail.

3.1 Emergency Use-Case

In the emergency use case, the corresponding interactive learning scenario targets at the estimation of the optimal height of barriers for flood control in a flood distribution scenario. Automatic parameter exploration of the barrier height parameter allows the efficient placement of barriers for specified emergency situations. Planning and design began on the basis of the information compiled in D4.1, and the use case was refined. A conceptual workflow was created during the design phase by defining the structure, shown in Figure 17.

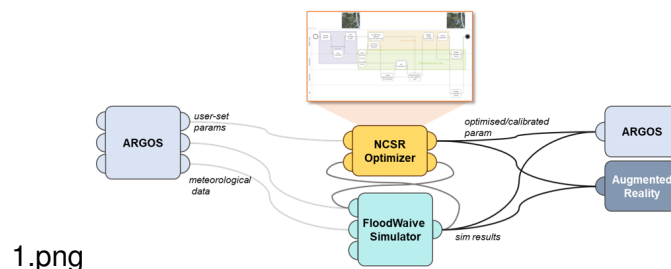


Figure 17: Conceptual workflow of the Emergency use-case interactive learning scenario for flood control.

In the basic function, the workflow is triggered in the ARGOS system, which sends a configuration file to a Kafka topic. The Kafka topic is read by the optimiser. In the message from the ARGOS system to the NCSR optimiser, the following information is written to the Kafka topic ‘UPB-CREXDATA-Flooding-Optimizer-Input’:

Kafka Topic: 'UPB-CREXDATA-Flooding-Optimizer-Input'

Data Format: json

Sample message:

```
1 {"area": "Inssbruck",
2  "optimizerRequestID": "38db3355-723d-4664-a844-29c95a76b034",
3  "pointOfInterest": {
4    "longitude": 11.386849096575194,
5    "latitude": 47.26622058072917},
6    "rainfallEvent": {
7      "name": "Event 1",
8      "rainfallIntervals": [ {"duration": 3600, "
9        intensity": 20} ] },
10     "fitnessScore": 0.3,
11     "barriers": [ {
12       "name": "barrier\_1",
13       "tag": "WALL",
14       "minHeight": 1,
15       "maxHeight": 4,
16       "geometry": {
17         "type": "Polygon",
18         "coordinates": [
19           [ [11.385359625045984, 47
20             .26571488109357],
21             [11.386242275854302, 47
22               .26528409571503],
23             [11.386242275854302, 47
24               .26529308893109],
25             [11.385359625045984, 47
26               .26572387430963],
27             [11.385359625045984, 47
28               .26571488109357]]
29         ]
30       }
31     },
32     {"name": "barrier\_2",
33       "tag": "WALL",
34       "minHeight": 1,
35       "maxHeight": 4,
36       "geometry": {
37         "type": "Polygon",
38         "coordinates": [
39           [ [11.387124924558218, 47.
40             26726942434718],
41             [11.387097342180798, 47.2
```



```

35         65995826051096] ,
36         [11.387110595026495,47.2
37         65995826051096] ,
38         [11.387138177403916,47.2
39         6726942434718] ,
40         [11.387124924558218,47.2
41         6726942434718]]
42     ]

```

The message contains all relevant information for running the optimizer and the FloodWaive flooding simulation. This includes the area, which can be either Dortmund or Innsbruck, and an individual optimiser ID to clearly identify the optimised calculation. The main condition is the point of interest (POI). The point of interest is defined using WGS84 system coordinates. The coordinates refer to a point that can be defined by a human expert in ARGOS. The RainfallEvent determines all parameters necessary to start the simulation. This includes rainfall duration and intensity as described in D2.3. The last part of the message contains all barriers drawn by the user. The number of barriers can vary from 1 to n. Each barrier must contain the following information: name, type of barrier (i.e. wall), minimum height, maximum height and geometry as a polygon, and thus the start and end points of the barriers, also in WGS-85 system coordinates. All the respective parameters are set by the user via the ARGOS user interface, and all calculations are performed using the respective workflow in AI Studio (see Figure 18).

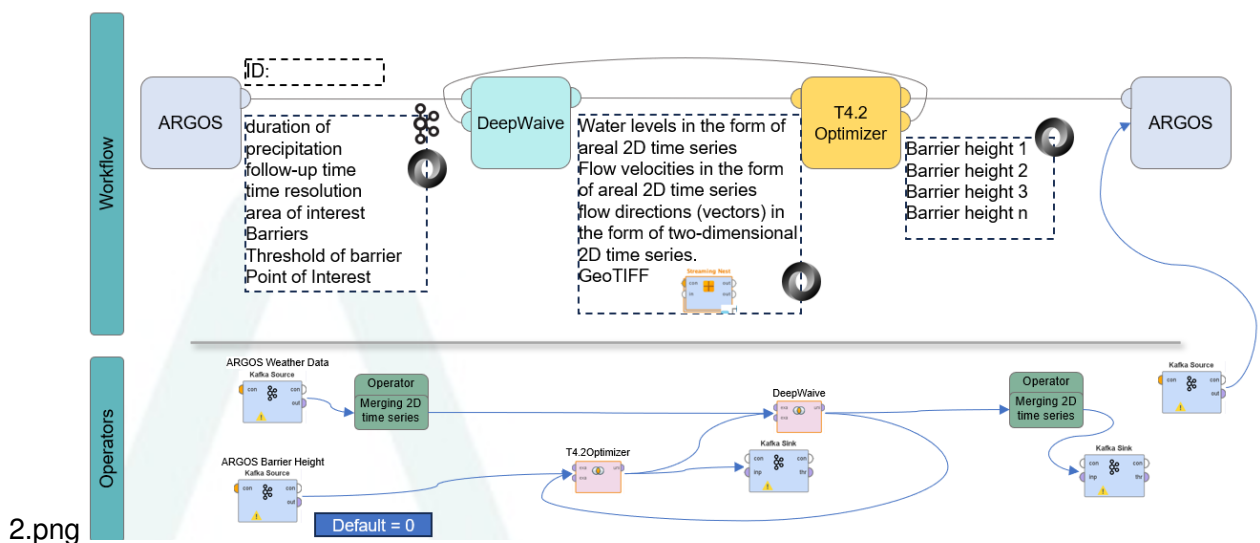


Figure 18: AI Studio workflow that implements the Emergency use-case interactive learning scenario for flood control.

The json message is uploaded via a command in the ARGOS system and sent to the

CREXDATA system via the Kafka topic 'UPB-CREXDATA-Flooding-Optimizer-Input'. The workflow is triggered in Rapid Miner AI Hub and starts the subsequent components. For the optimizer to begin its execution, a number of barriers as well as the point of interest (POI) must have been selected by the end-user. After the optimizer begins its operation, it queries the FloodWaive simulator and triggers a number of simulations. The server capacities at FloodWaive are parallel, and a simulation is started to evaluate the effectiveness of different barrier heights. Optionally, server capacities are automatically increased to reduce the computing time.

The simulation in FloodWaive with the different barrier heights is based on the previously defined rainfall event. The rainfall event does not vary here; what changes is the height of each barrier, according to the searching procedure of the optimization process, which in our case is a Genetic Algorithm. The overall aim is to find the minimum height of each barrier so that the POI does not flood, or the flood levels are kept low. In addition, a safety factor can be defined for the barrier height, which is always adhered to. The barrier height depends on the topological subsoil on which the model in FloodWaive is based.

The optimizer then triggers the simulation and iterates until it has defined the optimum. Once an optimum barrier height has been defined, the respective threshold value or safety factor for each barrier is maintained and the point of interest at the location defined by the user remains dry. The optimizer then sends a message to the Kafka topic 'UPB-CREXDATA-Flooding-Optimizer-Output'.

Kafka Topic: 'UPB-CREXDATA-Flooding-Optimizer-Output'

Data Format: json

Sample message:

```
1 {"optimizerRequestID": "c4e64cfa-5cc0-4fdf-b296-68e765bc1e6f",
2  "floodwaiveSimulationID": "simulation-50096bce-03d4-44ad-9688-
   aade6bb0d0a6",
3  "optimizedBarrierHeights": [
4      {"name": "barrier\_3",
5       "height": 10},
6      {"name": "barrier\_4",
7       "height": 11}
8  ]
9 }
```

The message format is also JSON and is monitored in real time by the ARGOS system. When a message is written to the topic, it is visualised directly in ARGOS as a result. The following values are transferred: (i) 'optimizerRequestID' to clearly assign the defined optimisation to a user, (ii) 'floodwaiveSimulationID' contains the final water distribution model and the (iii) 'optimisedBarrierHeights', which contains all optimised barriers. No coordinates are returned, only the name of the barrier with the respective optimised barrier height. The barrier height can vary from barrier to barrier and depends on the topological conditions of the terrain model. The 'floodwaiveSimulationsID' returns the final status. This means that the water distribution

is calculated over time and for each geolocated position. The ID is stored in a GeoTIFF file which can be accessed via the ARGOS system.

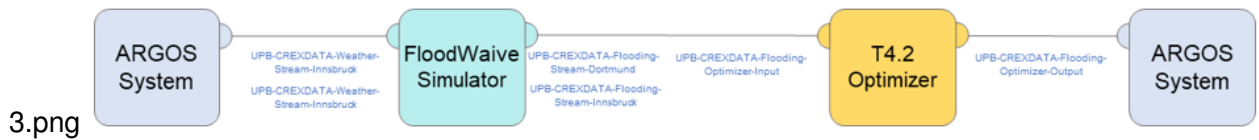


Figure 19: Kafka topic streams of the barrier height optimization.

All these components were used to develop custom operators in the AI Studio and AI Hub. Starting with the trigger in ARGOS, the following Rapid Miner workflow begins. After starting, the workflow in Figure 20 is launched.

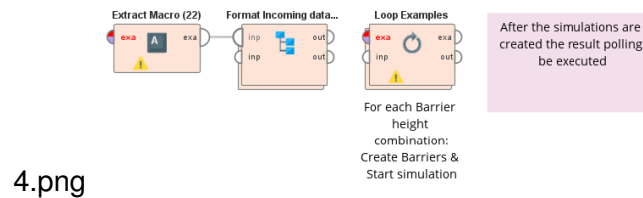


Figure 20: AI Studio workflow overview.

From there, read max water level and write to kafka is triggered as shown in Figure 21. This workflow starts the definition of the water level at the point of interest. This allows the optimization to protect the POI area from flooding.

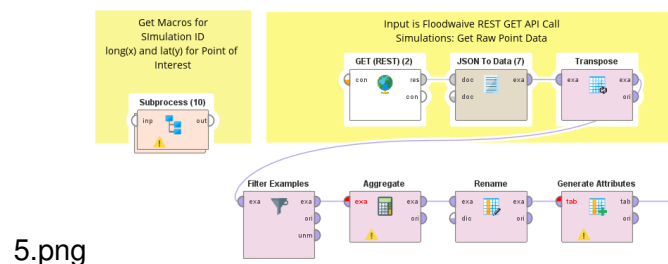


Figure 21: AI Studio workflow subprocesses.

Finally, the process is repeated until the optimum barrier height is reached, as can be seen in Figure 22.

The workflow ends with a final view of the optimised barriers and the json message in the corresponding Kafka topic. The Kafka topic is finally visualised by the ARGOS system, including all barriers, dry points of interest and barrier locations, as well as the displaced water distribution.

3.1.1 Barrier Height Optimization

To determine the best barrier height at the user-defined locations, we developed three optimization methods of three different algorithm categories. The first is a Genetic Algorithm (GA) [64, 73] that is a population-based evolutionary approach, the second is Simulated

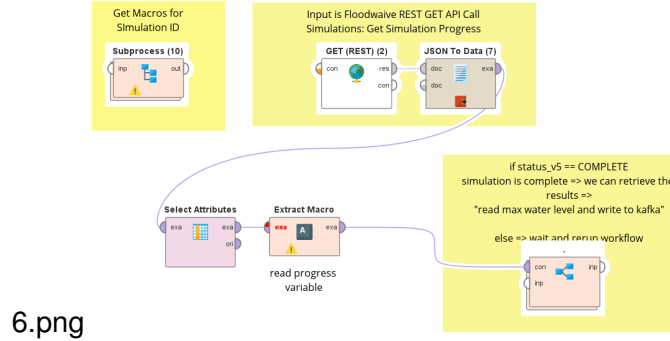


Figure 22: AI Studio workflow subprocesses.

Annealing (SA) [39, 73] that is a search metaheuristic optimizing a single individual—in contrast to the GA that operates on many, and the third is the Tree-structured Parzen Estimator (TPE) [64, 4] that utilizes kernel density estimates. In principle, all these three optimization methods search the barrier height parameter space to obtain a solution that minimizes the water height at a point of interest, while keeping the height of each barrier the minimum necessary to achieve this. These objectives are generally formulated as a function, but in the CREXDATA use-case we use the simulation results from Floodwaive.

As a pre-trial evaluation and comparison process, we constructed two objective functions that adhere to the flooding scenario. This dictates that for zero and near-zero barrier heights the value should be high, but as these increase then the value drops lower, ideally up to a global minimum, which corresponds to the shortest barriers that can protect the POI. After that point, although higher barriers are still able to protect the POI, the cost for installing them is unnecessary. Thus, the objective score starts again to increase proportionally. With this into consideration, we use a linear version with a single global minimum, and a more complex one with a large amount of local minima and a global one. Both these functions are defined in $\mathcal{R}^{n \times 1}$, where n is the number of user-set barriers under examination. For illustration purposes, we provide 3D plots for $n = 2$ in Figure 23 for the linear case, and in Figure 24 for the non-linear one.

3.1.2 Optimizer performance comparison

To compare the performance of the optimizers, we executed the algorithms repeatedly using the linear and the non-linear objective functions for different random seeds and for different numbers of water barriers placed, i.e. 3 and 6. Recall that in Figures 23 and 24 we illustrate the case for 2 barriers, and as we can see, all three algorithms perform quite well, managing to find barrier heights that result to acceptable solutions which adequately minimize the objective function. However, we also need to examine how the number of barriers affect algorithm performance. All optimizer cases were written in Python 3.9 and the experiments were executed on an 2,3 GHz 8-Core Intel Core i9 with 16GB of RAM. Hyperparameter tuning has been performed using smaller scale exploratory trials. All algorithm runs examined at least 7500 candidate barrier heights.

The results from 3-barrier and 6-barrier trials are shown in Table 6 and 8 for the linear function, and Table 7 and 9 for the non-linear one.

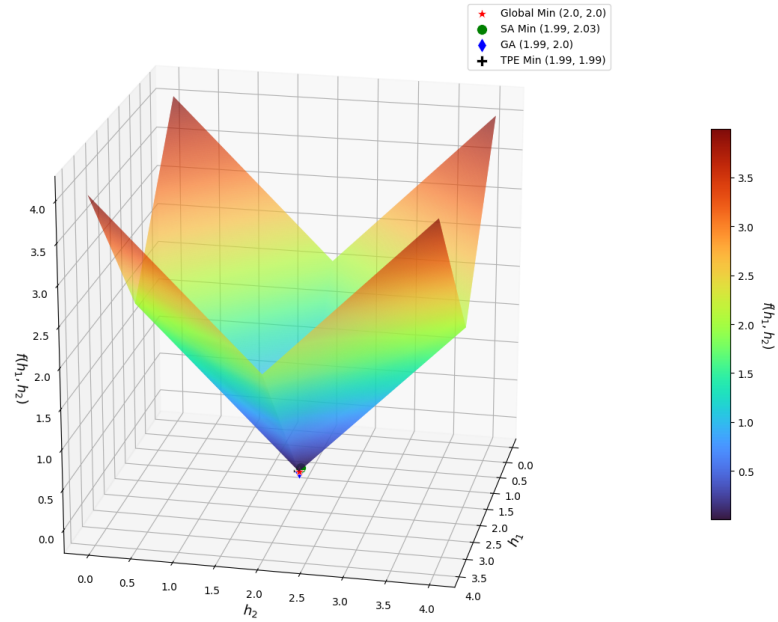


Figure 23: Linear objective function and championed points by each optimizer case.

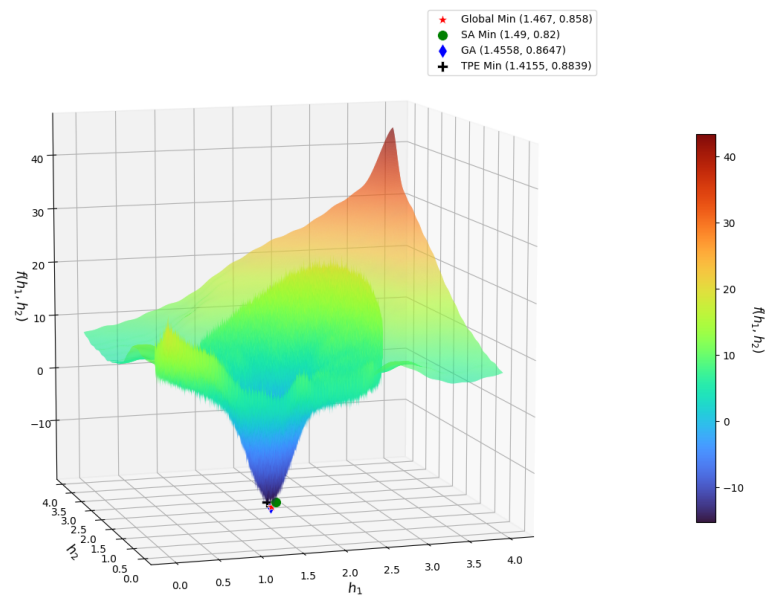


Figure 24: Non-linear objective function and championed points by each optimizer case.

Table 6: Comparative results for the linear objective function with 3 barriers.

Random Seed	SA Min.	SA Min. Index	GA Min.	GA Min. Index	TPE Min.	TPE Min. Index
321	0.4	6751	0.0	5929	0.12	2876
432	0.24	663	0.0	7175	0.14	6433
543	0.56	2716	0.0	5637	0.12	3109
654	0.4	4131	0.0	6955	0.09	5321

Table 7: Comparative results for the non-linear objective function with 3 barriers.

Random Seed	SA Min.	SA Min. Index	GA Min.	GA Min. Index	TPE Min.	TPE Min. Index
321	-13.18	4777	-17.61	4432	-14.94	5000
432	-13.09	4422	-17.73	4995	-13.97	3736
543	-13.09	3455	-17.19	6262	-15.51	5124
654	-13.18	7119 99	-17.23	7034	-14.36	2657

Table 8: Comparative results for the linear objective function with 6 barriers.

Random Seed	SA Min.	SA Min. Index	GA Min.	GA Min. Index	TPE Min.	TPE Min. Index
321	1.2	5659	0.0	7046	1.99	5832
432	0.99	5109	0.0	7187	2.21	6891
543	1.14	3250	0.0	6705	1.4	5352
654	1.45	5874	0.0	6673	1.36	5541

Table 9: Comparative results for the non-linear objective function with 6 barriers.

Random Seed	SA Min.	SA Min. Index	GA Min.	GA Min. Index	TPE Min.	TPE Min. Index
321	-15.23	4857	-17.46	7716	-14.36	4489
432	-15.61	929	-17.25	7189	-13.9	5957
543	-15.23	427	-17.07	6225	-14.61	1976
654	-15.26	5573	-17.6	7010	-13.98	4532

We can see that in all cases, GA manages to either find the minimum (linear case for both 3 and 6 barriers), or approximate it better than the SA and TPE (non-linear case for both 3 and 6 barriers). SA on the other hand, although not reaching as close to the minimum as GA, it manages to find acceptable approximations with less iterations, which might turn out to be useful in case of emergencies that are time-critical. Regarding time, SA is the fastest, returning the best barrier height configuration in less than a second, the second fastest is the GA with less than a minute running time (50 sec.), and the slowest turned out to be TPE with 200 sec., mainly due to the computationally demanding task of obtaining the kernel density estimates. Due to the robustness in the results of the GA, we chose to integrate it for use in the workflows for the real-world trials.

3.1.3 Workflow Evaluation & Lessons Learnt

Since the topic of interactive learning for simulation exploration would be too technical for end users of the CREXDATA system, the entire workflow described above was implemented using user interfaces in the demonstrator system of the weather emergency use case. The demonstrator system is described in detail in deliverables D2.2 and D2.3. The ARGOS system forms the basis for this and acts both as a data source (e.g., for retrieving weather data to configure flooding simulations) and as a data sink by visualizing the results of CREXDATA technologies, such as Interactive Learning for Simulation Exploration. This ensures that end users can make real use of the technology. Interactive Learning Technology was made available to users from the fire departments in Dortmund (Germany) and Innsbruck (Austria) in the final evaluations in test scenarios created for this purpose and tested. The following describes the process that an end user went through in the final evaluation, which is fully synchronized with the AI Studio workflow described above.

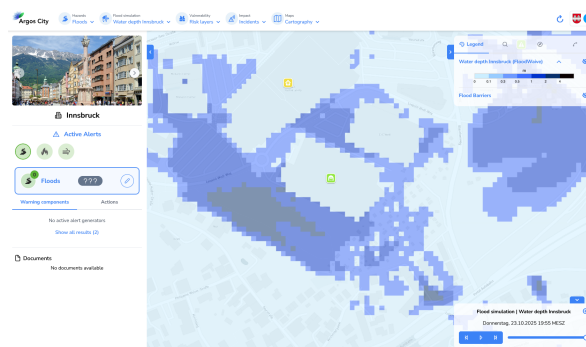


Figure 25: Initial ARGOS view.

First, an initial flood simulation is carried out to estimate the spread of water in the urban area based on current weather forecasts and precipitation predictions (Figure 25). Users then assess which areas are particularly worthy of protection or are likely to be severely affected. Barriers are placed at the inlets to these peripheral areas (Figures 26 and 27). In ARGOS, the user selects two corner points of a straight barrier, which are then connected to each other. The user repeats this process as many times as there are barriers to be erected (Figure 29). Once all barriers have been set, the “point of interest” is set on the map. The point of interest defines the area on the map and in the urban area where the water level is to be minimized by the use of barriers (Figure 28). The optimizer then calculates the optimal barrier height

as described above, which runs in the background from the end user's perspective. The parameter for the optimal barrier height is then output via the workflow and visualized for the end user together with the corresponding simulation result of the flood simulation in ARGOS. This allows the end user to make decisions such as how high the barriers need to be and how many emergency personnel are required to erect them.

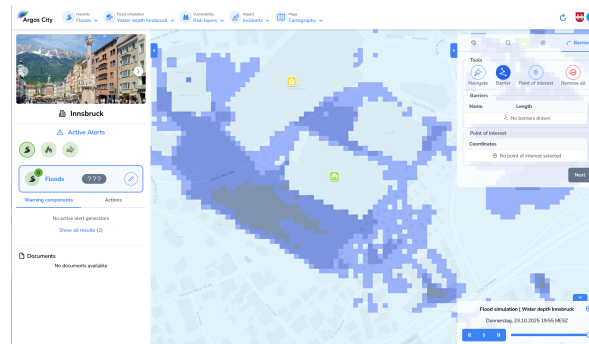


Figure 26: Opening the Barrier Menu.

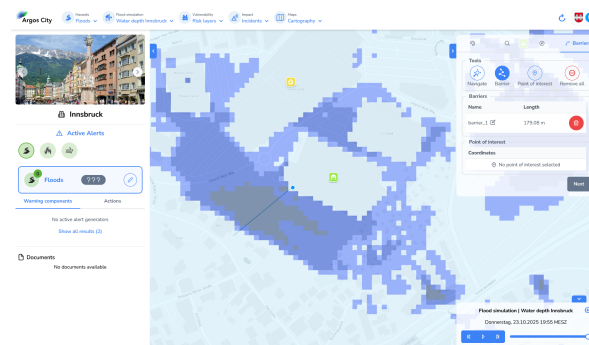


Figure 27: Setting a barrier location.

After using the interactive learning workflow, users evaluated the use case, relevance, and benefits of the technology in practice. In a questionnaire, users quantitatively rated the statement “Automatic parameter optimization (e.g., height of a mobile barrier) helps to use available resources and emergency personnel efficiently in practice.” on a rating scale from 1 to 10 (1 = strongly disagree; 10 = strongly agree). The end users’ overall assessment of the interactive learning technology was positive. Automatic parameter optimization was considered helpful in deploying resources and emergency personnel more efficiently in the field. The average rating across all locations was 7.64 points, with eleven users surveyed. This shows clear user approval of the usefulness of the technology. Users in Innsbruck rated the function slightly higher at 8.00 points than those in Dortmund at 7.43 points, which could indicate local differences in the framework conditions. In detail, this suggests that users see the added value of the technology particularly in its ability to support complex decisions under time pressure. By automatically optimizing parameters such as barrier height, the technology can help to use material and personnel availability more efficiently and avoid sources of error caused by manual calculations. It thus contributes to increased operational reliability and planning efficiency. This aspect is perceived as particularly important in dynamic situations, such as during flooding.

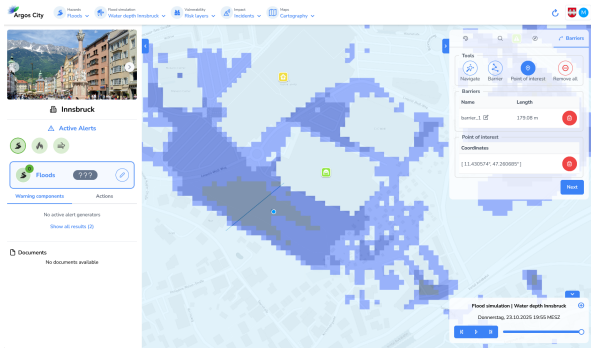


Figure 28: Setting the Point of Interest.

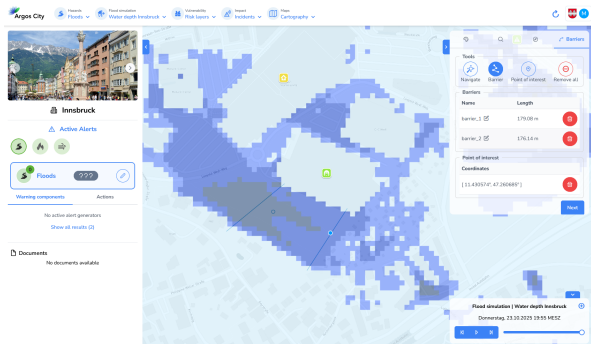


Figure 29: Setting an additional barrier location.

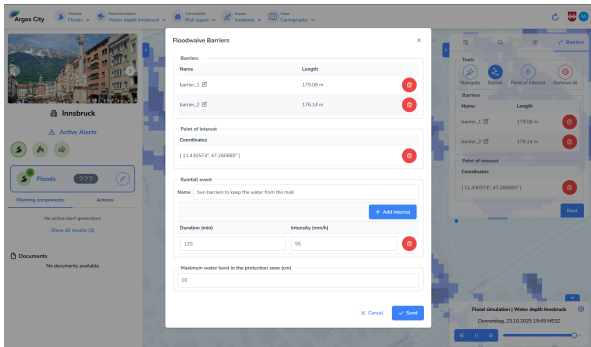


Figure 30: Defining simulation details.

The qualitative feedback also confirms this positive overall impression. Particular emphasis was placed on the intuitive and simple application, the uncomplicated handling, and the high added value of the real-time application for the task force. These points illustrate that, despite its technical complexity, the system is user-friendly and can be easily integrated into existing workflows. Users see quick access to decision-making information in real time as a decisive advantage in operational use. It was noted that an automatic suggestion function for the optimal positioning of barriers could further improve the technology. Such an enhancement would make the decision support even more comprehensive and further develop the system into a holistic optimization platform. Overall, the results prove that the interactive learning technology is practical, user-friendly, and effective. It supports emergency responders in making quick and informed decisions, thereby making a valuable contribution to optimizing operations.

3.2 Health Crisis Use-Case: Multi-scale Simulations

To support the development of adaptive and user-guided multiscale simulations, a dedicated interactive learning framework has been implemented within the multiscale scenario of the health emergencies use case. This framework enables experts to observe, analyse, and steer PhysiBoSS alveolar infection simulations through an integrated loop that combines real-time data visualization and direct user intervention. By linking outputs with intuitive control interfaces, the system allows researchers to guide simulation trajectories based on live biological metrics—such as cell-state counts and viral load—thereby transforming model exploration into a continuous, knowledge-driven process. The interactive learning setup connects three functional layers :

1. **Data Streaming and Monitoring (Kafka Layer):** PhysiBoSS simulations continuously produce quantitative outputs describing the evolving biological state of the alveolar compartment. These include the temporal evolution of live, apoptotic and necrotic cell populations, as well as viral load and cytokine concentrations. The results are streamed through Kafka and displayed in a web-based dashboard that updates dynamically at each simulation step, providing an intuitive and immediate overview of infection progression.
2. **User Intervention Workflow (RapidMiner / AI Studio Layer):** The human-in-the-loop logic is implemented through a dedicated process in Altair AI Studio (RapidMiner), as illustrated in Figure The workflow is composed of three key operators:
 - **Kafka Cluster:** establishes the connection to the distributed data streams produced by PhysiBoSS.
 - **Python Intervention Script:** provides a programmable node for user-defined interventions. Based on the observed simulation behaviour, the expert can modify model parameters in real time—such as introducing gene knock-outs (*MAPK14*, *FADD*, *AKT1*), adjusting drug concentrations, or altering immune-response thresholds when infection or cell-death rates surpass predefined levels.
 - **Write Kafka Topic:** transmits the chosen intervention as a structured JSON message to a dedicated topic (e.g. `pb_drug`) with low latency (1 s message interval). The PhysiBoSS orchestrator automatically consumes these control messages and

updates the configuration of the next simulation iteration.

This architecture converts the simulation workflow from a static batch process into a responsive feedback system where computational models and expert knowledge co-evolve.

3. **Visual Analytics and Feedback (Browser Layer):** The visual interface displays real-time plots of cell-state counts and environmental variables, allowing users to monitor the effects of each intervention. Updated simulation curves appear within seconds of user action, providing immediate quantitative feedback. When approximately 10% of the simulation time has elapsed, the ETSC module produces an early prediction of the likely outcome. As illustrated in Fig. 31, the same browser interface also enables the user to introduce drug or gene knock-outs directly during the simulation.



Figure 31: Browser-based dashboard used in the multiscale interactive learning workflow. The interface combines real-time cell-state trajectories, early ETSC predictions, and an intervention panel that allows the user to trigger drug or gene knock-outs during execution.

Interactive Learning Logic. Each human intervention is recorded as a structured triplet (*state*, *action*, *outcome*):

- **State:** the current physiological conditions derived from live-stream data (e.g. cell ratios, infection intensity);
- **Action:** the user-defined modification (gene knock-out, environmental perturbation);
- **Outcome:** the state of the alveoli after the simulation has finished given the intervention.

The interactive workflow inherently generates a structured dataset that captures each instance of user intervention together with the corresponding simulation context and outcome. This emerging dataset represents a valuable resource for the continuous evolution of the system, as it can support the analysis of decision patterns, the quantification of intervention effects, and the identification of biologically meaningful parameter regions. In future extensions

of the framework, these data could be leveraged to train reinforcement or active learning components capable of autonomously suggesting new interventions based on previous user experience. This would enable the interactive learning approach to progressively evolve toward a hybrid human–AI decision-support paradigm, strengthening both the explainability and efficiency of multiscale simulation exploration.

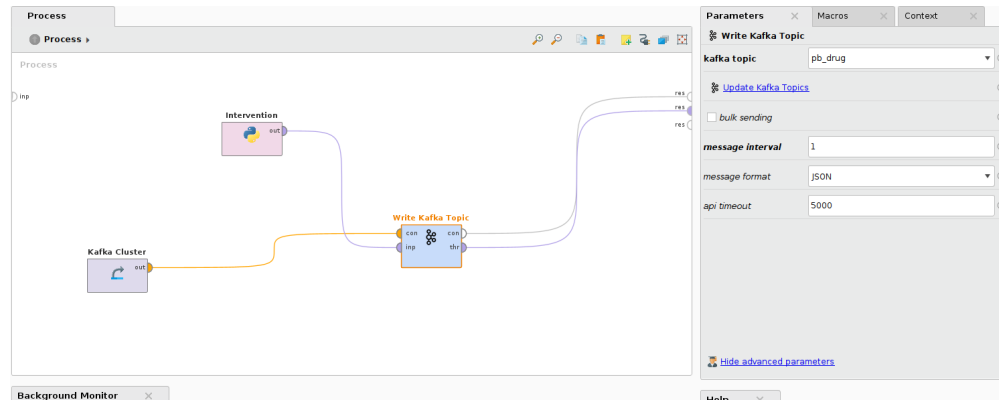


Figure 32: Human-in-the-loop intervention workflow in RapidMiner. The process connects the Kafka Cluster (input data stream) to a Python-based intervention node and a Write-Kafka operator that publishes JSON control messages (pb_drug). These updates are consumed by the PhysiBoSS orchestrator while a browser-based dashboard visualises real-time cell counts, enabling experts to steer infection dynamics interactively.

3.2.1 Quick Outcome Characterization via Early Time-series Classification

In D4.1 we implemented a benchmarking framework for ETSC algorithms that was used to determine the best option among a number of available approaches. Having compared algorithm performance on a dataset produced by PhysiBoSS, we determined that the most advantageous among them was TEASER [68].

This algorithm transforms input time-series to words using WEASEL, and the measured frequencies of these words are forwarded to a logistic regression classifier. The first prefix is of size equal to the length of the time-series divided by S , and the last one is the full time series. For each prefix, a WEASEL-logistic regression pipeline is trained and then used to obtain probabilistic predictions. These predictions are passed on to a One-Class SVM, uniquely trained for each prefix length using only the correctly classified data by the pipeline. If the One-Class SVM accepts the prediction, i.e. it is marked as belonging to the correct class, then the last criterion to generate the final output is the consistency of the particular decision for v consecutive prefixes. The parameter v can be selected during the training phase by performing a grid search over the set of values $\{1, \dots, 5\}$. For each candidate value, the method tries to classify all the training time series, and the one that leads to the highest harmonic mean of earliness and accuracy is finally selected.

To apply it in the Multiscale Simulations case, we adopted the WEASEL+MUSe [67] transformation that supports multivariate input, which in terms of PhysiBoSS data consists of the

following variables: (i) the total number of cells, (ii) the total number of Epithelial cells, (iii) the Alive Epithelial cells, (iv) the Apoptotic Epithelial cells, (v) the Necrotic Epithelial cells, and (vi) the Infected Epithelial cells. The implementation are made in Python and the source code is available in a public repository² along with execution instructions. It can be configured to either train on a user-specified dataset, or use a pre-trained model to generate class label predictions directly, omitting the training step. During the prediction phase, the input is received in real-time via a Kafka topic in the form of a json string containing the variable values at each time step. When the algorithm is confident enough to generate a prediction, it writes to a different Kafka topic the estimated class label. This in turn can be further consumed by other components in the workflow.

The ETSC implementation of CREXDATA has been used to characterize the outcome of health interventions under consideration by an end-user of the workflow, before the full completion of the respective simulations, thus aiding to the saving of time and computational resources.

3.2.2 Evaluation

The multiscale infection scenario integrates high-resolution airflow simulations from Alya with cellular-level infection dynamics from PhysiBoSS. Within this context, UC2a introduces a capability that was not previously available in PhysiBoSS-based workflows: the ability to interactively explore, monitor, and steer simulation experiments through the Interactive Learning framework developed in WP4. This functionality represents an important methodological advance, as conventional PhysiBoSS simulations run sequentially and in batch mode, without any mechanism for intervention during execution or immediate visual assessment of how changes in parameters affect model behaviour.

The new Interactive Learning workflow provides an environment where simulation inputs can be modified, re-launched, and immediately analysed from a graphical interface. This allows both experienced modelers and domain scientists to observe the impact of parameter changes, including modifications in the Boolean regulatory network, without needing to wait for full simulation completion. Such early and continuous feedback is particularly valuable in agent-based models, where small changes in the regulatory logic may alter the dynamics of viral replication, immune activation, or cell fate decisions. For the end user, this represents a significant increase in transparency and ease of experimentation: rather than manually altering configuration files or analysing results post-hoc, users can explore intervention strategies, test hypotheses, or examine parameter sensitivities interactively, reducing the barrier to entry for non-technical audiences.

For model developers, this approach considerably accelerates the validation cycle. Previously, adjusting a rule in the Boolean model or modifying a microenvironmental parameter required executing a full simulation and analysing the output in external tools. Now, the impact of such modifications can be observed in near real time, allowing the modeler to identify unintended behaviours, confirm expected trends, or refine model structure more efficiently. This reduces the turnaround time for model development and enables a more iterative and exploratory modelling process.

In parallel, the Extreme-scale Time Series Classification (ETSC) algorithm has been evaluated

²TO-DO: Fill in the repository URL

as a method for predicting the final outcome of a PhysiBoSS simulation from early-trajectory information. Applied to the multiscale infection scenario, ETSC offers a mechanism to infer the likely evolution of infection severity or tissue outcome long before the simulation completes. This capability has the potential to save a substantial amount of computational time and resources, especially in scenarios involving large parameter sweeps or patient-specific simulations where hundreds or thousands of model instances are required. By identifying simulations whose outcomes are predictable early on, ETSC enables early termination of redundant runs and prioritises computational effort on those cases that truly require detailed analysis.

Taken together, these two components—Interactive Learning and ETSC—introduce a new level of agility to the multiscale infection modelling workflow. Interactive Learning accelerates the exploration and understanding of model behaviour from the perspective of the user, while ETSC accelerates the computational pipeline itself by reducing full simulation load. Both capabilities extend the functionality of PhysiBoSS beyond what is possible in its original form.

3.2.3 Lessons Learned

The development and evaluation of Interactive Learning and ETSC within the multiscale infection scenario provided several insights relevant to both technical integration and practical utility. One of the clearest observations is that modelers and end users often want to explore hypotheses or test rule modifications without waiting for long running simulations to complete. In such situations, predictive models such as ETSC offer substantial value by providing an early indication of whether a simulation is likely to evolve toward a particular outcome. This reduces both experimentation time and computational overhead and supports a more iterative approach to model development and scenario exploration.

From an infrastructure perspective, the initial integration attempts exposed limitations imposed by MareNostrum 5 network isolation, which prevents direct communication between external tools and HPC compute partitions. As described in Deliverable 2.3, these restrictions were successfully mitigated through the use of a dedicated virtual machine, allowing the workflow to bypass HPC communication constraints while still supporting the data transfer patterns needed for interactive monitoring and predictive evaluation. This solution provided a stable pathway for implementing the Interactive Learning loop despite architectural constraints of the HPC system.

An additional lesson concerns the broader applicability of predictive approaches such as ETSC beyond this specific multiscale scenario. The ability to infer the likely outcome of a simulation from partial trajectories can be generalised to other agent-based or continuous models that share similar dynamical structures. In many modelling tasks, early identification of stable, diverging, or trivial dynamics can prevent unnecessary simulation effort and focus computational resources where they are most informative. Thus, while ETSC was applied to COVID-19 lung infection simulations, the underlying methodology can support other CREX-DATA scenarios and more generally any modelling workflow involving large-scale parameter exploration.

Finally, the introduction of interactive monitoring and configuration manipulation shifts the modelling process toward a more iterative and exploratory mode. This type of interactivity encourages hypothesis-driven experimentation and enables rapid feedback cycles, aligning

simulation practice more closely with the needs of both scientific investigation and decision-support contexts. The combination of interactive mechanisms with predictive acceleration illustrates how AI-assisted simulation workflows can significantly improve the usability, efficiency, and interpretability of complex multiscale models.

3.3 Health Crisis Use-Case: Epidemiological Scenario

As part of the health crisis use case, **episim-emews** [55] was developed: a high-performance exploratory workflow that integrates the **EpiSim.jl** simulator with the **EMEWS** framework for model exploration and optimization. This workflow enables the parallel execution of multiple instances in a high-performance computing (HPC) environment. It supports both large-scale parameter sweeps and, for optimization problems, guided parameter space searches through the integration of evolutionary algorithms.

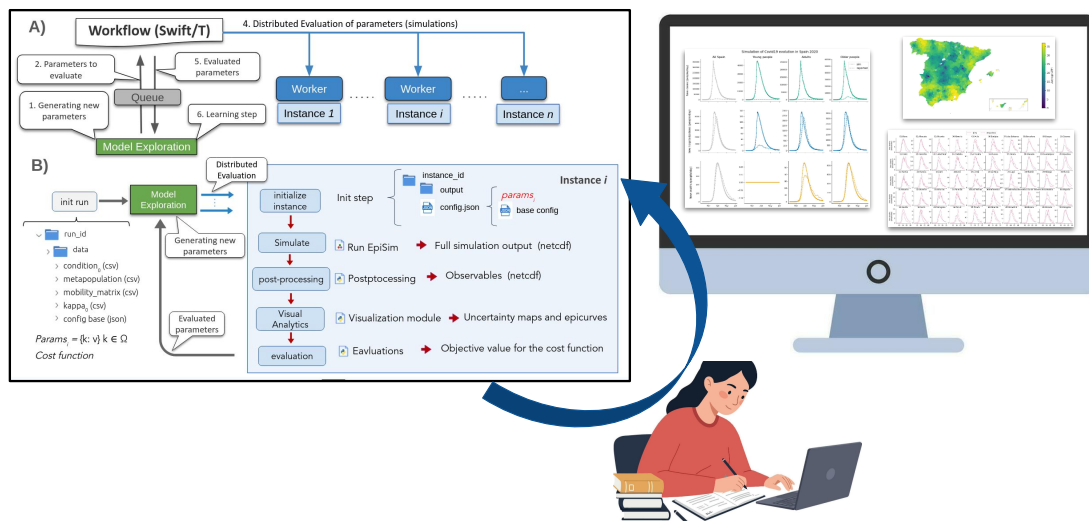


Figure 33: Overview of the interactive learning workflow coupled with episim-emews. The figure illustrates the main steps of the episim-emews workflow along with the interactive learning process enabled by user-driven analysis between successive experiments

Each execution cycle follows six main steps:

1. **Initialization:** The workflow initializes the experiment by loading input data (population, mobility matrix, initial conditions) and generating configuration files for each instance.
2. **Distributed execution:** Parameter sets are dispatched to HPC worker nodes, each running an independent EpiSim simulation in parallel.
3. **Simulation and post-processing:** Each worker executes the model and processes the outputs into standardized observables (e.g., incidence, hospitalizations), which are stored in NetCDF file.
4. **Visualization:** Optional plotting and visual analytics functions generate intermediate results such as epidemic curves or uncertainty maps to facilitate dynamic analysis.

5. **Evaluation** : The system computes a **cost function** comparing simulated and real data, quantifying model fit and serving as the objective metric for optimization.
6. **Learning step** : Evaluated parameters are fed back to the exploration algorithm which updates the parameter set for the next iteration based on the optimization strategy (e.g., GA or CMA-ES)

The workflow is fully customizable and supports uncertainty assessment using visual analytics tools developed in collaboration with T5.2, enabling interactive learning. Between each round of experiments, users can analyze previous results to refine input files (e.g., initial conditions or parameters intervals), select which parameters to estimate based on the objective (e.g., calibration or policies evaluation) and choose how to compute the cost function used to measure the fitness of each simulation. Additional workflow features can also be configured through a JSON file, which defines all the steps executed during a model exploration run, like how outputs are processed and aggregated, whether to generate instance plots, and which parameter exploration algorithm to employ.

3.3.1 Reinforcement Learning

The **EpiSim.jl** simulator was also coupled with reinforcement learning techniques to develop **episim-rl** [54], a workflow designed to identify optimal strategies for dynamic non-pharmaceutical interventions. In this workflow, the agent is trained by iterating over episodes of the same epidemic scenario, in this case corresponding to the first wave of the COVID-19 pandemic.

Since the policies learned by the agent need to be analyzed and interpreted by human experts, and also since the state-action space is of manageable size, we opted for a Q-learning [14] algorithm instead of a deep neural networks-based one (e.g. DQN [65]). Ease of implementation and lower computational requirements also aided to this selection.

Figure 34 illustrates the overall episim-rl workflow, showing how the reinforcement learning module and the epidemic simulator are coupled. Each simulated episode is divided into intervals (e.g., two weeks), during which a specific confinement policy is selected by the agent. The simulator then initializes the epidemic system and runs the corresponding interval to evaluate the chosen strategy. After each interval, the workflow collects the current state of the simulation (e.g., ICU stress, disease severity) and computes a reward function that captures the epidemiological outcomes, like hospitalizations and deaths, as well as the socio-economic costs. This information is fed back to the agent, which uses it to assess the impact of the confinement strategy and inform its next decision. By iterating over many episodes, the agent gradually learns a set of intervention strategies that minimize negative health outcomes while accounting for socio-economic impact.

After each interval simulation, the reinforcement learning module receives feedback from the simulated environment to support its learning process. The simulator output is processed to extract the current state and compute the reward function.

The **state** is determined by the week just evaluated, the policy applied during that interval, and several quantities that describe the epidemic status. These parameters quantify the level of ICU occupancy (*ICU stress*), the incidence (*disease spread*), the total number of deaths during the interval (*disease severity*) and the average number of people that one infected will

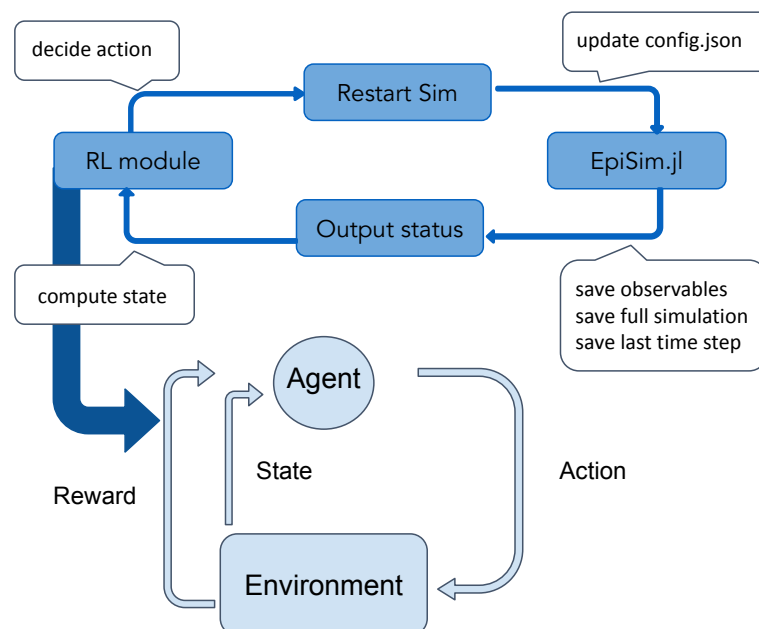


Figure 34: Diagram of the episim-rl workflow, illustrating the integration of reinforcement learning techniques into EpiSim.jl. The process begins by restarting the simulation and updating the configuration file with the new NPI parameters (applied policy). EpiSim.jl is then executed, the outputs are collected and sent to the RL module, which updates the agent and selects a new action, restarting the cycle

infect in a fully susceptible population (*basic reproductive number*, R_0). These attributes are discrete, and the combination of all their possible values defines the state space.

The **reward function** decreases with higher levels of ICU occupancy, deaths, and stricter confinement measures, increasing the penalty of strict policies as the epidemic outcome improves. In this way, it discourages actions that lead to poor epidemic outcomes while avoiding unnecessary interventions, thereby balancing health and socio-economic impacts.

The **actions** available to the agent are limited to a total of 121 possible combinations. Policies are defined by two parameters: κ_0 , representing the mobility reduction and δ , representing the reduction of number of contacts. Both parameters can take values from 0 to 1 in increments of 0.1, where 0 indicates no restrictions and 1 represents maximum confinement.

3.3.2 Evaluation & Lessons Learned

Interactive Learning

The interactive learning workflow was tested through parameter calibration to reproduce the dynamics of the COVID-19 pandemic in Spain. The first experiments were run using data from previous research led by Alex Arenas et al [5]. Although the results were promising, there was room for improvement.

Initially, the simulator was able to reproduce the total daily deaths curve almost perfectly, and the hospitalizations trends were also well captured. However the incidence data was less reliable due to significant underreporting. The age-stratified dynamics showed some discrepancies, particularly among younger populations, and a more detailed analysis revealed larger discrepancies at the provincial level.

Through an interactive learning approach combined with visual analytics, the calibration was progressively refined, as shown in figure 35. First the parameter intervals were broadened, since there was observed no major effect on overall epidemic dynamics and enabled new solutions. Next, the parameters were allowed to vary independently by age group, removing predefined dependencies, and achieving a better fit of the age-stratified total curves. The parameter controlling the hospitalization rate among young individuals was also introduced to the calibration, as this curve exhibited the largest mismatch. Up to this point, the fit of age-specific curves had improved considerably, but some discrepancies remained in the provincial-level dynamics.

These residual mismatches suggested that the initial conditions were not fully accurate due to underreporting. To address this, the initial conditions were iteratively adjusted, progressively adding infection seeds and verifying that the province-level death curves converged toward the observed data.

These iterative refinements showcase the effectiveness of the interactive learning workflow in improving model calibration through guided exploration and user feedback. By combining data-driven adjustments with expert interpretation, the workflow achieved a more accurate and consistent reproduction of the COVID-19 dynamics across age groups and provinces.

Reinforcement Learning

The episim-rl tool was assessed to identify optimal confinement strategies for the first-wave

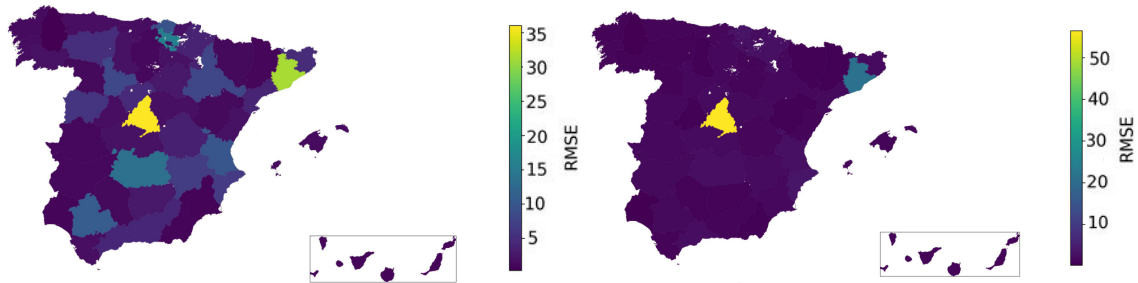


Figure 35: Maps of the root mean square error (RMSE) across provinces for the first and last experiments, illustrating the improvement of the calibration.

episode of the COVID-19 pandemic in Spain. These policies are modeled through two parameters: κ_0 , representing the mobility reduction and δ , representing the reduction of number of contacts. To this end, we designed three scenarios of different severity, allowing the agent to explore the full parameter space.

The first scenario (good) corresponds to implementing confinement measures as soon as the first cases were detected. The second scenario (medium) represents the real timing of the applied measures, while the third scenario (bad) simulates a delayed response, in which interventions are implemented one month after the real policies took effect. To enable comparison among the three scenarios, all initial conditions were rescheduled to start on March 15.

For this experiment, 5000 episodes were run sequentially, with a scenario randomly selected for each episode. Figure 36A) shows the convergence of the total reward over the training episodes, indicating that the agent reaches an optimal strategy. Several confinement strategies successfully minimize the spread of the pandemic while considering the socio-economic costs. Figure 36B) shows how the optimal strategy varies with the severity of the epidemic, while improving in all cases the final outcome compared with the actual evolution of the COVID-19 pandemic in Spain. The measures become stricter as the epidemic worsens: in the good scenarios, policies remain relatively mild, while maintaining control of the epidemic, whereas in the medium and bad scenarios, stronger measures are required initially to reduce the epidemic outbreak. In the bad scenario, policies are later relaxed, while in the medium scenario they intensify toward the end, likely preventing a second peak.

These results demonstrate that episim-rl can effectively learn adaptive confinement strategies that balance epidemiological and socio-economic objectives. The workflow successfully identifies optimal policies under different epidemic severities, highlighting its potential as a decision-support tool for managing future outbreaks.

Lessons Learned

The implementation of interactive and reinforcement learning techniques proved instrumental in enhancing both the accuracy and interpretability of the demonstrators. The interactive learning workflow enabled an iterative calibration process that not only improved the model fit but also revealed inconsistencies and missing information in the underlying data. Addressing these gaps proved essential, as incomplete or underreported data—such as underestimated incidence or inaccurate initial conditions—can significantly affect the reliability of epidemic

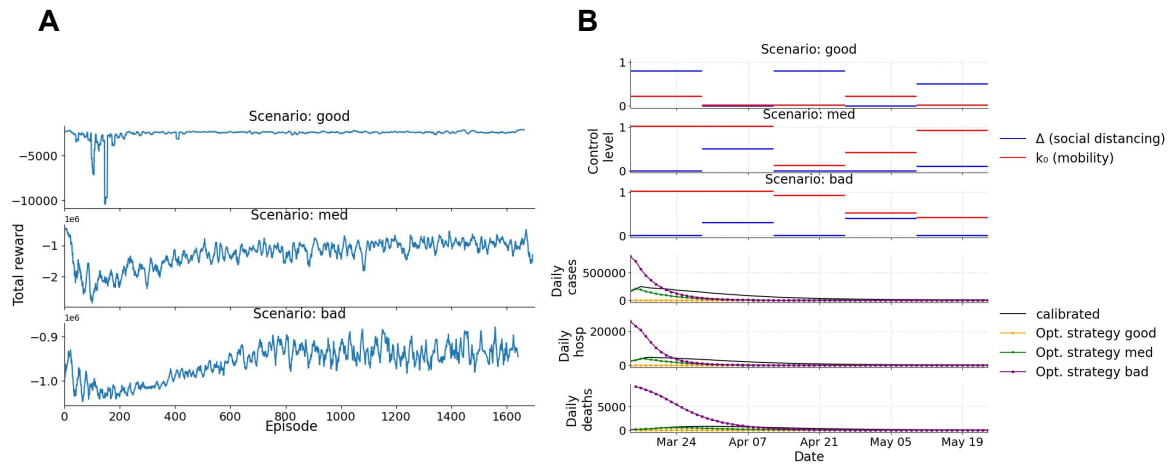


Figure 36: A): Convergence of the total reward over the training episodes for the three scenarios. B): Top episode from each scenario, showing at the top the confinement policies applied every two weeks and, at the bottom, displaying the simulated curves for cases, hospitalizations, and deaths. These are compared with the calibrated simulation that represents the real outcome, illustrating improved results.

simulations. Through interactive learning and visual analytics, it was possible to iteratively refine parameters and initial conditions, improving the overall consistency between simulated and observed dynamics.

Reinforcement learning, in turn, allowed for the systematic exploration of alternative confinement strategies that would have been difficult to evaluate manually. The experiments demonstrated the potential of the approach to identify adaptive, balanced intervention policies, though they also highlighted some limitations. In particular, further runs or extended training episodes could help the agent achieve stronger convergence and more stable policy performance, especially when exploring complex or noisy epidemic scenarios.

Overall, integrating these AI-driven methods into the EpiSim framework significantly improved the robustness and explanatory power of the simulations. Interactive learning helped uncover and correct data mismatches, while reinforcement learning expanded decision-support capabilities, enabling the discovery of optimized, data-informed intervention strategies. These experiences underline the importance of combining human expertise, data quality assessment, and machine learning to effectively guide epidemic modeling and policy evaluation.

3.4 Maritime Use-Case

3.4.1 Interactive Learning Workflow

The Maritime Use Case aims to develop integrated solutions for the early detection, forecasting and solution planning of critical maritime events in order to support safe navigation and informed decision-making for key stakeholders and decision makers in maritime operations. Building on the requirements elicitation for maritime complex critical events described in

Deliverable D2.1, two complementary modules have been designed that address different risk factors at sea:

- **Collision forecasting and rerouting:** This component combines historical and real-time navigation data with predictive models to forecast potential vessel collision scenarios. When a risk is detected, it generates optimised alternative routes that minimise the probability of an incident while maintaining operational efficiency.
- **Hazardous weather rerouting:** This module ingests meteorological and oceanographic forecasts to identify segments of a planned route that are likely to be affected by severe weather conditions. It then suggests route adjustments that avoid hazardous areas without incurring significant delays or deviations from common traffic routes.

In this context, interactive simulators have been developed to enable maritime operators to explore alternative solutions and optimize vessel navigation parameters under the collision avoidance and hazardous weather routing use cases. The simulator can ingest simulated (i.e., artificially generated) vessel positional data and allows users to adjust critical operational parameters in order to evaluate different navigation strategies. All simulated events and recommended actions are presented through a graphical user interface (GUI), facilitating systematic evaluation and iterative refinement of the decision-support algorithms.

3.4.2 Vessel Collision Avoidance Interactive Simulator

The collision avoidance simulator constitutes a key component of the maritime decision-support environment. Development work over the reporting period finalised both the backend simulation functions and the web-based front end, which has been deployed as a dedicated microservice hosted on the AWS infrastructure of Kpler and integrated within the distributed CREXDATA architecture based on the Pekko framework. This component provides operators with two interactive prediction capabilities:

1. **Route-step simulation:** Using forecasts of future vessel positions and probabilistic collision points, the simulator computes a sequence of waypoints that redirect vessels around the predicted collision point while satisfying safety and compliance constraints. The resulting step-wise path is visualised on a nautical chart, enabling the operator to comprehend how the vessel(s) will maneuver to avoid the potential incident.
2. **Speed-based rerouting:** In scenarios where a collision is forecasted at a considerable distance, adjusting vessel speed may be preferable to a full course change. A speed simulation function therefore allows the operator to specify alternative target speeds for the involved vessels. The system then recalculates the trajectories under the new speeds and presents the resulting collision avoidance paths. This functionality is particularly useful when the forecasted collision point lies outside the current safety radius and early speed adjustments can preempt a course change.

During the simulation the user interface concurrently overlays short-term vessel traffic flow forecasts, produced by data-driven models running on the CREXDATA platform. This overlay highlights areas of high traffic density, thereby assisting the operator in choosing alternative routes that avoid not only the forecasted collision point but also congested regions (see Fig. 37). A separate panel allows operators to set the target speeds and initiate the rerouting algorithm.

Figure 37 illustrates the collision avoidance simulator graphical user interface. Two vessels are shown approaching a predicted collision point; the system computes a safe collision-avoidance path (green polyline) and displays the forecasted collision location together with the underlying traffic flow forecast. The speed input panel enables the user to enter target speeds in knots and trigger the rerouting function. Figure 38 demonstrates the effect of the speed simulation function: when the operator selects target speeds of 15 kn and 10 kn respectively for the two vessels, distinct simulated collision-avoidance paths are generated. These examples show how the simulator supports proactive decision making through adjustable parameters and real-time visual feedback.

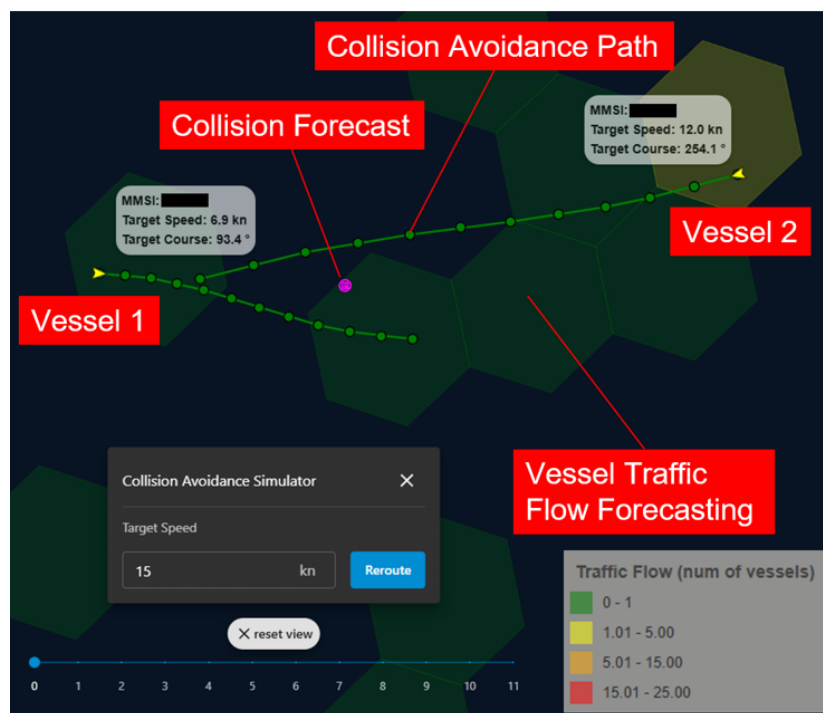


Figure 37: Collision avoidance simulator interface. The display shows two vessels approaching a forecasted collision point, the computed collision-avoidance path (green), the collision forecast location (purple marker), and an overlay of vessel traffic flow density. The control panel allows the operator to set the target speed and reroute.



Figure 38: Output of the speed-based simulation function. In this example the operator specifies target speeds of 15 kn (red) and 10 kn (blue) for the two vessels as the forecasted collision positions lies too far ahead. The simulator computes new collision-avoidance paths (yellow-coloured paths) that divert the vessels around the forecasted collision point.

3.4.3 Hazardous Weather Routing Interactive Simulator

To support safe and efficient maritime navigation under adverse meteorological conditions, a comprehensive hazardous weather routing simulator has been developed and deployed as part of the CREXDATA platform. The underlying simulation functions have been finalised and exposed as dedicated application programming interfaces (APIs), allowing multiple routing alternatives to be explored through a unified front end. The simulator integrates real-time and forecasted meteorological data to propose navigational strategies that mitigate exposure to hazardous weather while respecting voyage efficiency constraints.

The following interactive capabilities are provided:

1. **Departure time simulation:** Users may specify alternative departure times for a voyage in order to postpone departure until a hazardous weather system has moved through the planned track or to depart early and outrun an approaching storm. This step is aligned with typical procedures undertaken during voyage planning by vessel crews in order to avoid arriving at their destination port during an extreme weather event or in cases where avoiding severe weather conditions would result in a significant deviation from the common traffic or the shortest route to the destination.
2. **Speed simulation:** By adjusting the vessel's cruising speed, operators can evaluate whether slowing down or accelerating will avoid intersection with forecasted severe weather conditions. This functionality supports decisions such as reducing speed to allow a storm front to pass ahead and avoid a significant route deviation or increasing speed to reach shelter before conditions deteriorate.

3. **Alternative Route simulation:** The system can compute alternative routes towards the destination based on the common traffic patterns and the shortest route. Each candidate route may be assessed for safety and efficiency, offering a trade-off between voyage length and exposure to adverse conditions in comparison to the weather optimized route. This step allows vessel crews to assess the necessity of selecting the weather optimal route and the trade offs with operational efficiency.
4. **Port-to-port route:** Given an origin and destination port, the simulator provides a set of candidate routes that balance route length, vessel performance and weather conditions. This is useful for voyage planning and schedule optimisation in liner shipping.
5. **Point-to-port route:** When a vessel is already at sea, this option allows operators to plan a safe course from the current position or any position of choice to a chosen port, for example to seek shelter during a storm.
6. **Point-to-point simulation:** For reaching a specific positions to wait for orders, seek shelter outside of port areas or for maritime operations such as search and rescue or scientific surveys, users can plan a route between arbitrary coordinates on the map and avoid hazardous weather conditions.

Figure 39 illustrates the hazardous weather routing simulator GUI. The map displays the Western Mediterranean with shaded regions representing forecasted weather risk. The simulator panel on the right exposes separate generators for the *traffic-optimized* route, the *shortest route* and the *departure time and speed simulators*. These options allow the operator to compute a traffic-aware alternative (blue dashed line), the shortest geodesic route (brown dotted line) or a weather-optimized path (green solid line) that aims to avoid areas with adverse weather and follow paths with optimal weather conditions. The GUI also provides fields to set the departure date and time, specify the vessel's speed in knots and select origin and destination ports (e.g., Algeciras to Ponza). Weather forecast layers can be selected via an overlay control, and the time slider at the bottom of the display offers an interactive means of examining how meteorological conditions evolve along the proposed routes.

These capabilities are critical in a variety of maritime operations. In cargo shipping, for example, a container vessel departing Algeciras for the Italian port of Ponza Figure 39 may need to adjust its departure time or route to avoid a forecasted severe weather system on route; the simulator may recommend diversion depending on the severity and timing of the storm. Passenger ferries operating across the Mediterranean can use the speed simulation to determine whether slowing down to let a storm pass or accelerating to outrun it will minimise passenger discomfort and fuel consumption. Research vessels and offshore service ships can plan point-to-point transits that minimise exposure to hazardous seas when repositioning between survey sites. When a vessel at sea must seek shelter due to deteriorating weather, the point-to-port simulator enables rapid evaluation of alternative safe ports and corresponding routes.

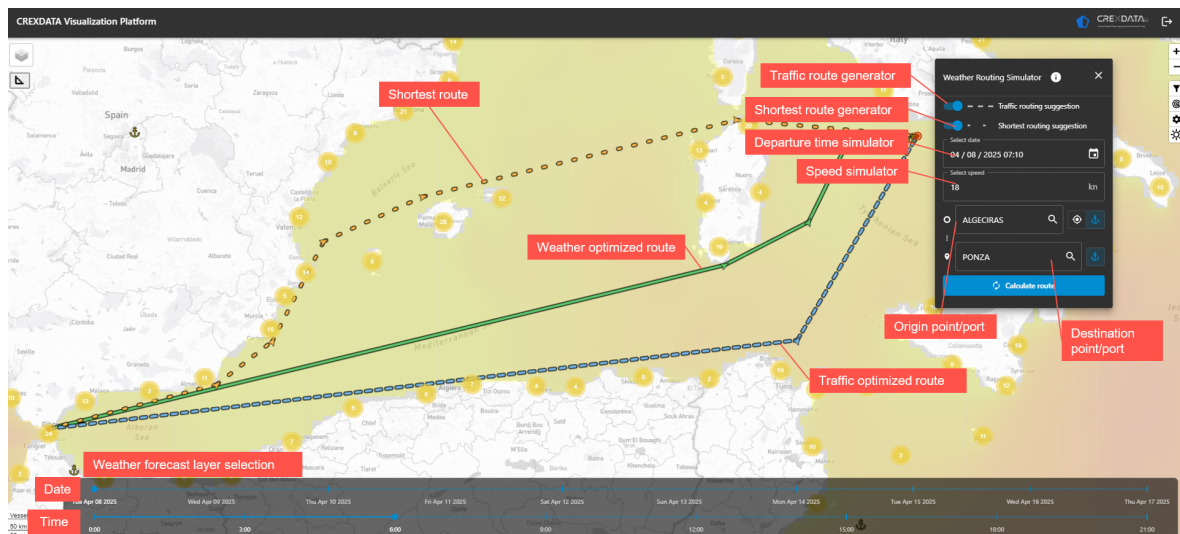


Figure 39: Hazardous weather routing simulator interface using the updated labeled screenshot. Weather risk is visualized as shaded regions over the Western Mediterranean. The interface distinguishes between a traffic-optimised route (blue dashed line), a weather-optimised route (green solid line) and the shortest route (brown dotted line). Operators can choose among dedicated route generators (traffic, shortest) and simulation functions for departure time and speed, define the origin and destination ports, and select weather forecast layers. The timeline at the bottom enables exploration of the temporal evolution of the forecast along each route, and numeric markers indicate the severity of weather events in each sector.

The Hazardous Weather Routing Simulator functionality is enabled both as an API for user defined inputs and for AIS positions processed on stream by Pekko. Figure 40 presents the example view for a vessel from the real time AIS stream from MarineTraffic. Hazardous weather routes are generated for all vessels (AIS messages) on the MarineTraffic stream provided that the destination port is correctly defined in the static part of the AIS message and matches with a port entry in the MarineTraffic Registry.

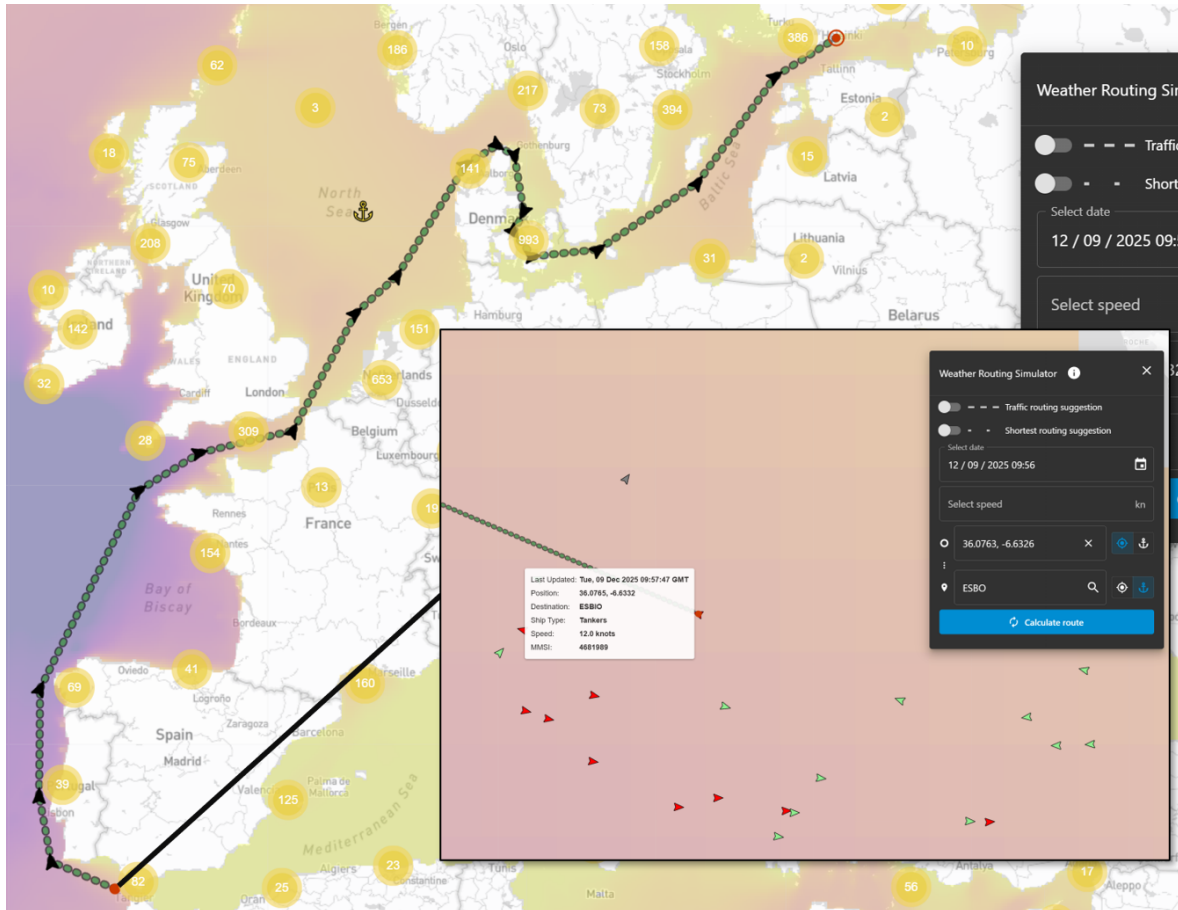


Figure 40: Hazardous weather routing simulator interface on Pekko processed AIS positions. Users are able to select a vessel based on its last reported AIS position from the MarineTraffic AIS stream and view its projected weather route towards its reported destination port in real-time. Users may also directly access the simulator functions for that vessel directly through the simulator window to evaluate alternative travel speeds and routes.

3.4.4 Evaluation & Lessons Learnt

A series of experiments were undertaken to assess both the qualitative behaviour of the developed simulators and the quantitative performance of the weather-routing algorithm. For the hazardous weather routing component, forty origin–destination queries were executed across the Gulf of Mexico and U.S. East Coast during the hurricane-intense month of September 2022. As summarised in Table 5, routes generated by our method were on average only 3.3 % longer than the baseline shortest paths, yet they reduced very extreme weather penalties by 35 % and extreme penalties by 15 %, demonstrating that the algorithm successfully steers vessels away from the most hazardous areas while maintaining operational efficiency. A large-scale performance test comprising 1200 randomly sampled route requests across the North Atlantic showed that the system’s response time increases linearly with route length but remains well below four seconds even for trans-oceanic voyages (Fig. 16). These results confirm that the fusion of AIS-derived traffic patterns with dynamically updated weather forecasts

yields practical, real-time routing recommendations that balance safety and efficiency.

The development of the collision avoidance and hazardous weather simulators has also provided qualitative insights. The collision avoidance simulator, now integrated into the MSA visualisation platform, enables operators to explore alternative rerouting options by adjusting vessel speed or course in real time; preliminary trials indicate that the interface presents clear visual cues and generates updated collision-free routes instantaneously, supporting rapid decision-making. The hazardous weather simulator offers complementary functions—such as departure-time and speed simulations, and port-to-port, point-to-port and point-to-point routing—that allow users to evaluate multiple strategies for avoiding storms, with a user interface that visualizes weather layers and candidate routes. Together, the two simulators illustrate how a combination of historical fleet intelligence and forecast data can underpin an integrated maritime decision-support system. Lessons learnt during the evaluation highlight the importance of high-resolution weather data, robust interpolation on the H3 grid, and adaptive weighting between traffic and weather penalties. Future work will extend these concepts by incorporating vessel-specific characteristics, fuel-efficiency models and reinforcement learning to support even more nuanced, autonomous voyage optimization.

4 Federated Machine Learning

4.1 Introduction

This task delivered a suite of communication-efficient algorithms for Federated Learning (FL) as part of the CREXDATA project. The developed methods enable decentralized machine learning across multiple data owners—such as organizations, sensors, or edge devices—without requiring the exchange of raw data.

A central outcome of our work is the ability to reduce communication overhead—which is the main bottleneck in federated setups—by orders of magnitude without adverse side effects. We achieved this by introducing adaptive, data-driven mechanisms that allow each participant to decide locally and autonomously when to engage in communication, based on key metric indicators. This dynamic scheme forms the core of our proposed algorithm, *Federated Dynamic Averaging (FDA)* [78].

At Deliverable D4.1 and at [78] we demonstrated that our FDA algorithm can outperform the state-of-the-art algorithms for Federated Learning by 1-2 orders of magnitude in terms of reduced communication overhead. We also demonstrated how this technique can be incorporated within NeRF computation algorithms, allowing visualizing scenes interactively in 3D, such as the ones required at the Weather Emergencies Use Case. In this deliverable, we make several important contributions:

- Given that FDA depends on a threshold parameter that needs to be tuned, we provide a modified version of FDA, termed FDA-OPT (Sections 4.2- 4.3), where this parameter is dynamically adapted for a non-expert user who has no experience with a specific federated learning task. We demonstrate that FDA-OPT outperforms state-of-the-art algorithms, even at a non-tuned disadvantageous setting for it.
- To allow the user to estimate the impact of parameter settings under a federated learning application with different connectivity settings, before actually deploying a federated learning algorithm, we integrate the NS3 discrete-event network simulator (Section 4.5) with our federated learning techniques.
- Given the very slow computation of existing NeRF techniques, we design (Section 4.6) novel, scalable and efficient NeRF techniques for rapid adaptation during an emergency situation.
- All our algorithms can now operate either standalone as a service (Sections 4.4, 4.5 and 4.6), or through a graphical workflow designed at RapidMiner's AI Studio. The integration with AI Studio is detailed in Deliverable D4.2.

4.2 FDA-OPT: Generalized Federated Dynamic Averaging

In this section, we propose FDA-OPT (depicted in Algorithm 1), the unified generalization of both FDA and FEDOPT, with a **Dynamic Variance Threshold** option. Specifically, we introduce a novel dynamic mechanism that automatically calibrates the variance threshold during training, eliminating the need for any manual tuning. The highlighted lines of the algorithm represent the core additions that distinguish FDA-OPT: namely, the logic for

Table 10: Overview of the FEDOPT and FDA-OPT families of algorithms. Depending on CLIENTOPT and SERVEROPT a different name is derived for the FL algorithm. FDA-OPT is our proposed algorithmic family, which will be introduced in Section 4.2.

FEDOPT	FDA-OPT (ours)	CLIENTOPT	SERVEROPT
FEDAVG [52]	FDA-SGD [79]	SGD	SGD (lr = 1.0)
FEDAVGM [30]	FDA-SGDM [79]	SGD	SGDM [75]
FEDADAM [63]	FDA-ADAM [79]	SGD	ADAM [37]
FEDADAMW [88]	FDA-ADAMW [79]	SGD	ADAMW [45]
FEDADAGRAD [63]	FDA-ADAGRAD [79]	SGD	ADAGRAD [19]

monitoring model variance and dynamically terminating rounds. These additions act as an adapter to the original FEDOPT framework (non-highlighted lines), effectively generalizing it with an dynamic round termination scheme.

Operational Details. The algorithm begins with a user-provided local training duration τ . This value is expected to be based on well-established and empirically effective configurations from prior FEDOPT work. Then, FDA-OPT modifies this internally by extending it to a larger value $\tilde{\tau} \gg \tau$ (e.g., $\tilde{\tau} = 10 \cdot \tau$; see Line 2).

From this point onward, the algorithm proceeds using the standard round-based approach. At the beginning of each round t , a cohort of clients S_t is sampled. These clients train locally for up to $\tilde{\tau}$ steps (Lines 3–9). At the end of training—whose actual length may vary—we compute the “pseudo”-gradient from model changes and update the global model accordingly (Lines 18 and 21). This process is repeated for a total of T rounds. What sets FDA-OPT apart is its variance monitoring logic.

Specifically, during local training, clients periodically synchronize compact state information (Lines 11–12) to approximate the model variance (Line 13). However, querying variance too frequently (e.g., after every local step) can completely bottleneck a real-world FL system. This is not due to communication volume (the local state states are small), but due to the synchronous nature of the operation. To mitigate this, we introduce a configurable set of step indices, $\mathcal{I}_{\text{query}} \subseteq [1, 2, \dots, \tilde{\tau}]$, which determines the exact points where variance queries are performed (Line 10). In contrast to the original FDA formulation [78], which used $\mathcal{I}_{\text{query}} = [1, 2, \dots, \tilde{\tau}]$ (i.e., checking after every step), our method leaves this set user-defined. In practice, we encourage it to be significantly sparser. As we will show later, in our setup the variance is queried only once per epoch.

Lastly, we allow the threshold value, Θ_t , to evolve across training rounds. Rather than using a fixed value, the threshold is updated dynamically based on the observed training dynamics. Specifically, we define a function `THRESHOLDADJUST`, which takes as input the current threshold Θ_t , the actual variance value at the end of the round, and the final step index s_t where the round terminated (Lines 19–20). Intuitively, these inputs capture the key information needed to answer two questions: (1) *was the current threshold too high or too low?* and (2) *did the round terminate at a reasonable point (e.g., $s_t = 1$ should raise concerns)?* In the next section, we will explore practical choices for this function. Notably,

Algorithm 1 FDA-OPT

Input: Initial w_0 ; CLIENTOPT, SERVEROPT; Total rounds T ; Local training steps τ , specifically tuned for FEDOPT [63]

```

1: Set  $\Theta_0 = -\infty$ 
2: Choose extended  $\tilde{\tau} \gg \tau$  ▷ FDA-OPT extends local training
3: for each round  $t = 0, \dots, T - 1$  do
4:   Sample a subset  $S_t \subseteq \mathcal{K}$  of clients
5:   Set  $w_{t,0}^{(k)} = w_t$  for all  $k \in S_t$ 
6:   for each client  $k \in S_t$  in parallel do
7:     for each local step  $i = 1, \dots, \tilde{\tau}$  do
8:       Compute a gradient estimate  $g_{t,i-1}^{(k)}$  of  $\nabla F_k(w_{t,i-1}^{(k)})$ 
9:        $w_{t,i}^{(k)} = \text{CLIENTOPT}(w_{t,i-1}^{(k)}, g_{t,i-1}^{(k)})$ 
10:      If  $i \in \mathcal{I}_{\text{query}}$  then ▷ Query variance now
11:        Construct local state  $\mathbf{A}_{t,i}^{(k)}$  following Section 4.2.3
12:        Aggregate states  $\mathbf{A}_{t,i} = \frac{1}{|S_t|} \sum_{k \in S_t} \mathbf{A}_{t,i}^{(k)}$ 
13:        Variance approx.  $\nu_{t,i} = H(\mathbf{A}_{t,i})$  ▷ See Section 4.2.3
14:        If  $\nu_{t,i} > \Theta_t$  then ▷ Threshold violation
15:           $s_t = i$  ▷ Round's final step index
16:          break ▷ Training round terminates
17:         $\Delta_{t,st}^{(k)} = w_{t,st}^{(k)} - w_{t,0}^{(k)}$  ▷ Local model change
18:         $g_{t,st} = -\frac{1}{|S_t|} \sum_{k \in S_t} \Delta_{t,st}^{(k)}$  ▷ "Pseudo"-gradient
19:         $\nu ar_t = \text{Var}(\mathbf{W}_{t,st})$  ▷ Compute actual variance
20:         $\Theta_{t+1} = \text{THRESHOLDADJUST}(\nu ar_t, \Theta_t, s_t)$ 
21:       $w_{t+1} = \text{SERVEROPT}(w_t, g_{t,st})$ 
22: return  $w_T$ 

```

computing the variance in Line 19 requires no additional work, as it reuses quantities—the drifts and “pseudo”-gradient—that are already available (Lines 17–18).

4.2.1 Default Configuration for FDA-OPT

We adopt a conservative approach in the configurations below. These are the exact settings used in our experiments, as they yield stable and reliable performance across all tasks. While we believe there is room for improvement—potentially through more aggressive configurations (e.g., a larger linear coefficient of increase of τ)—we leave such tuning as a direction for future work, beyond the scope of this study.

Local Training Extension. We set $\tilde{\tau}$ to be twice the original τ plus a large constant. This constant is chosen to be eight times the size of a typical client dataset—denoted by e . Formally:

$$\tilde{\tau} = 2 \cdot \tau + 8 \cdot \lceil e \rceil, \quad \text{where} \quad e \approx \frac{1}{|\mathcal{K}|} \sum_{k \in \mathcal{K}} |\mathcal{D}_k| \quad (9)$$

Variance Query Indices. Within each round, we query the variance once per epoch. This completely alleviates the synchronization bottleneck encountered in the original FDA (as

detailed in Deliverable D4.1). We define:

$$\mathcal{I}_{\text{query}} = \{e, 2e, \dots, \lfloor \tilde{\tau}/e \rfloor \cdot e\}$$

THRESHOLDADJUST. In general, variance within a round t tends to increase as local training progresses. We set the threshold Θ_t such that it is approximately equal to the expected variance around the midpoint of local training (i.e., at step $\lfloor \frac{\tilde{\tau}}{2} \rfloor$). Assuming linear growth, we can estimate the midpoint variance of the round $t + 1$ using a simple proportion, based on known values from the current round t —specifically, the actual variance var_t and termination step s_t computed in Lines 15 and 19 of Algorithm 1, respectively:

$$\begin{array}{ccc} \text{var}_t & \xrightarrow{\text{after}} & s_t \text{ local steps} \\ \text{var}_{t+1}^{(est)} & \xrightarrow{\text{after}} & \frac{\tilde{\tau}}{2} \text{ local steps} \end{array}$$

With this linear prediction, we set the next round's threshold to the estimated variance at the midpoint of local training, $\text{var}_{t+1}^{(est)}$:

$$\Theta_{t+1} = \frac{\tilde{\tau}/2}{s_t} \cdot \text{var}_t$$

4.2.2 Discussion

FDA-OPT: Intuition. The main intuition for FDA-OPT is summarized in making the decision to synchronize dynamic, based on *model variance* during training. This metric is designed to capture the collective state of the training process. In what follows, we provide intuition on why this is the case. It is important to remember that the global model $\bar{w}_t$ and, by extension, the global drift $\bar{\Delta}_t$, are ultimately what we care about and evaluate.

Model variance is the difference between the average of the squared local drifts $\frac{1}{K} \sum \|\Delta_t^{(k)}\|_2^2$ and the squared global drift $\|\bar{\Delta}_t\|_2^2$. The first term reflects how far the individual worker models have moved—essentially, how much each worker has learned. The second term indicates how much of this learning is retained in the global model after aggregation.

The interplay between these two quantities is crucial. For example, when the local drifts are high but the global drift is low, the variance increases, signaling the need for synchronization. This scenario suggests that while individual workers have made significant progress (as indicated by high local drifts), this progress is not being effectively captured in the global model (indicated by the low global drift). In other words, the worker models have moved significantly, but the global model has remained relatively stationary in this high-dimensional space. This misalignment indicates that training is no longer progressing optimally, as the workers are moving towards disparate and conflicting local minima, making it crucial to synchronize and realign them. Conversely, when both the local and global drifts are either low or high, synchronization is not necessary, and the variance naturally remains low.

Neither the average of the local drifts nor the global drift alone provides a complete picture of the collective training progress. Relying solely on one or the other would lead to suboptimal synchronization decisions and likely prove ineffective. In FDA, it is the relationship between these quantities, as captured by the model variance, that offers valuable insights and guides the crucial decision of when to synchronize.

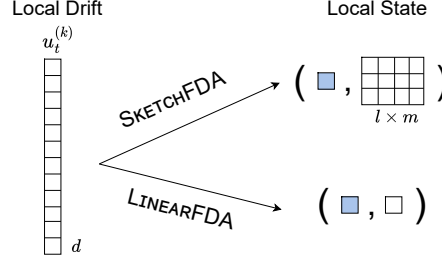


Figure 41: SKETCHFDA & LINEARFDA: Local State structure.

4.2.3 SKETCHFDA vs. LINEARFDA

Both methods send the squared norm of the drift $\|\Delta_t^{(k)}\|_2^2$, but differ in the additional accompanying lower-dimensional representation they transmit (Figure 41):

1. **SKETCHFDA**: An AMS sketch of the local drift. We employ AMS sketches $\text{sk}(\Delta_t^{(k)}) \in \mathbb{R}^{l \times m}$ as a low-dimensional representation of the local drifts $\Delta_t^{(k)}$.

Theorem 4. Let $l = \mathcal{O}(\log \frac{1}{\delta})$ and $m = \mathcal{O}(\frac{1}{\epsilon^2})$. Define the local state as

$$\mathbf{A}_t^{(k)} = \left(\|\Delta_t^{(k)}\|_2^2, \text{sk}(\Delta_t^{(k)}) \right) \in \mathbb{R} \times \mathbb{R}^{l \times m}$$

and the approximation function as

$$H(\bar{\mathbf{A}}_t) = \frac{1}{K} \sum_k \|\Delta_t^{(k)}\|_2^2 - \frac{1}{1+\epsilon} \mathcal{M}_2 \left(\frac{1}{K} \sum_{k=1}^K \text{sk}(\Delta_t^{(k)}) \right).$$

Then, the condition $H(\bar{\mathbf{A}}_t) \leq \Theta$ implies $\text{Var}(w_t) \leq \Theta$ with probability at least $(1-\delta)$ [78].

2. **LINEARFDA**: The dot product of a vector and the local drift. In this approach, instead of an AMS sketch, each local state contains the scalar value $\langle \xi, \Delta_t^{(k)} \rangle \in \mathbb{R}$, where $\xi \in \mathbb{R}^d$ is a unit vector, known to all workers.

Theorem 5. Define the local state as

$$\mathbf{A}_t^{(k)} = \left(\|\Delta_t^{(k)}\|_2^2, \langle \xi, \Delta_t^{(k)} \rangle \right) \in \mathbb{R} \times \mathbb{R}, \quad \|\xi\|_2 = 1$$

and the approximation function as

$$H(\bar{\mathbf{A}}_t) = \frac{1}{K} \sum_{k=1}^K \|\Delta_t^{(k)}\|_2^2 - \left| \frac{1}{K} \sum_{i=1}^K \langle \xi, \Delta_t^{(k)} \rangle \right|^2$$

Then, the condition $H(\bar{\mathbf{A}}_t) \leq \Theta$ implies $\text{Var}(w_t) \leq \Theta$ [78].

The key difference between these two variants lies in the fidelity of approximation of the model variance. While both methods conservatively overestimate the variance, SKETCHFDA provides a provably accurate estimation, which is expected to lead to fewer synchronizations.

LINEARFDA requires less computational effort and bandwidth to create and communicate the local states, but may overestimate variance by too much, causing unnecessary synchronizations.

SKETCHFDA: Choice of l and m . We empirically measured the approximation achieved with sketch dimensions of $l = 5$ rows and $m = 250$ columns: these settings yield an error bound of $\epsilon \approx 6\%$ and a probabilistic confidence of $(1 - \delta) \approx 95\%$. Based on our experiments, we have adopted these values in our experiments and recommend them. Using these values, the byte-size of a sketch is $l \cdot m \cdot 4 \text{ bytes} = 5 \text{ kB}$, significantly smaller than the size of all our models.

4.3 Experiments & Analysis for FDA-OPT

In this section, we outline the methodology and rationale behind evaluating the proposed FDA-OPT family of algorithms against FEDOPT. Specifically, we detail the infrastructure, datasets, data partitioning strategies, and models used in our study.

4.3.1 Setup

Infrastructure. We conduct our experiments using the *transformers* library from Hugging Face [89] with PyTorch [60]. The code for our implementation is available at GitHub³ All experiments are performed on a local cluster equipped with 2 NVIDIA A10 GPUs and 46 Intel(R) Xeon(R) Silver 4310 CPUs.

Tasks & Datasets. Natural Language Understanding (NLU) tasks are a core component of evaluating machine learning models in NLP and in Task 4.5. These tasks test a model’s ability to understand, reason, and infer relationships between pieces of text. To conduct our experiments, we select six widely used datasets and tasks from the GLUE [83] benchmark: MRPC [18], SST-2 [71], RTE [83], QNLI [83], MNLI-m and MNLI-mm [87]—each corresponding to a distinct NLU task. Since the GLUE test sets are unpublished, we follow prior studies [93] and use the validation sets as the new test sets.

Data Partitioning. A critical aspect of FL is the distribution of data across clients, which is characterized by high heterogeneity. Most NLP datasets are designed for centralized machine learning settings. In this subsection, we describe how we partitioned these datasets for FL.

Clients. The scale of data federation is determined by the number of clients, which generally falls into two categories: *cross-silo* and *cross-device* [32, 62]. In cross-silo FL, the number of clients is small—such as hospitals or financial institutions—often allowing all clients to participate in each training round. For this setting, we use the MRPC and RTE datasets. In contrast, cross-device FL involves a much larger pool of clients, such as mobile or IoT devices, where only a fraction of them can participate in each round. For this setting, we use the SST-2, QNLI, and MNLI datasets. Table 11 summarizes the client partitioning and sampling strategies across these datasets.

³<https://github.com/altairengineering/crexdata-public/tree/main/CREXDATA%20System%20Components/Federated%20Learning%20%28TUC%29/FDA-Opt-Sys>

Table 11: Client Partitioning Strategies per Dataset

	MRPC	RTE	SST-2	QNLI	MNLI-m	MNLI-mm
#Clients	10	10	100	250	1000	1000
Sample Size	10	5	10	10	10	10
Cross-Silo	✓	✓	✗	✗	✗	✗
Cross-Device	✗	✗	✓	✓	✓	✓

Table 12: Best-known accuracy scores (↑) for fine-tuning each model on each task in an ideal centralized setting. These values serve as target performance baselines

RoBERTa [44]				DeBERTa [29]	
MRPC↑	RTE↑	SST-2↑	QNLI↑	MNLI-m↑	MNLI-mm↑
90.2%	78.7%	94.8%	92.8%	90.6%	90.7%

Label Distribution. One of the most common heterogeneity scenarios in FL is the presence of non-IID label distributions, where clients possess data with disproportionately distributed labels. For instance, one client may predominantly have samples of label A, while another may mostly contain label B. This phenomenon frequently occurs in real-world applications and is modeled using the Dirichlet distribution [40, 42, 62]. The degree of label imbalance is controlled by the concentration parameter α . Larger values of α result in more uniform distributions, with $\alpha \rightarrow \infty$ producing a perfectly uniform distribution. Conversely, as α approaches zero, the distributions become highly imbalanced, with each client predominantly holding samples from a single label. In our experiments, we set $\alpha = 1.0$, introducing a considerable level of heterogeneity.

Models & Quality Baselines. In our experiments, we use the pre-trained transformer-based models RoBERTa [44] and DeBERTaV3 [29]. More specifically, we use the `roberta-base` and `microsoft/deberta-v3-base` checkpoints available through Hugging Face [89], which contain 125 million and 86 million parameters, respectively. Such encoder models are well-suited for FL, as they can be readily fine-tuned for a wide range of practical downstream NLP tasks [42] (e.g., sentiment analysis, textual entailment, etc.). We fine-tune both models with the FEDOPT and FDA-OPT algorithms.

Of course, our ultimate goal in FL is to train accurate and robust models. Thus, as in most of the FL literature [63], we evaluate algorithms based on how efficiently they manage to do that. In other words, instead of comparing final accuracy scores or communication alone, we ask: *how many communication rounds does it take to train a model with a specific target quality?*

To define those targets in a consistent and task-agnostic way, we express them as percentages of the best-known centralized performance. Given both models' widespread adoption, their top-reported performance when fine-tuned in centralized settings is well-documented (see Table 12). Naturally, due to the adverse conditions of data federation, FL performance is typically lower than centralized baselines. For example, the highest accuracy ever reported for RoBERTa when fine-tuned on MRPC is 90.2%; then, a target of 90% means attaining at least 81.18% accuracy ($90\% \cdot 90.2 = 81.18\%$).

Algorithms & Hyper-Parameters. We evaluate the most widely used algorithms from the FEDOPT family and compare their performance with our proposed FDA-OPT algorithms. To ensure a fair, apples-to-apples comparison, each FEDOPT algorithm is paired with its corresponding FDA-OPT variant using identical parameter configurations. For example, we compare FEDADAMW with FDA-ADAMW, keeping all optimizer settings fixed. The algorithmic pairs we compare are outlined in Table 10.

Our goal is to conclusively demonstrate that FDA-OPT outperforms FEDOPT and that hyper-parameters proven effective for FEDOPT not only work for FDA-OPT but also yield better results. To show this, we first find the optimal hyper-parameter configuration for each FEDOPT algorithm⁴. Then, we apply these to our proposed FDA-OPT counterparts. Given this approach, evidence that FDA-OPT outperforms FEDOPT is particularly meaningful.

Lastly, unless otherwise specified, we set the number of local steps τ equal to one epoch—that is, $\tau = \lceil e \rceil$ from Equation (9)—as this is the most common and widely accepted choice in the FEDOPT literature [42, 63, 30]. Moreover, we use a batch size of 8 and train for an exhaustive number of rounds, $T \in \{100, \dots, 1000\}$, ensuring that training continues well beyond likely convergence.

4.3.2 Results

In this section, we evaluate the performance of FDA-OPT against FEDOPT across a variety of FL tasks, focusing on two key aspects: communication efficiency and convergence behavior, for the same model quality. Notably, our FDA-OPT algorithms operate under configurations optimized for their FEDOPT competitors—which makes the following observed improvements all the more noteworthy.

Communication-Efficiency. The communication overhead incurred by each algorithm is directly analogous to the number of rounds. For FEDOPT, communication between clients and the server occurs only at the end of each round. In FDA-OPT, there is additional communication due to the transmission of small sketches. However, the size of these sketches is negligible compared to the overall communication cost of transmitting language models (it is $\times 10^6$ smaller).

As is standard in the literature [84], we evaluate our algorithms by comparing their performance against predefined target metrics—such as accuracy ($\uparrow$). The primary objective of the algorithms is to train a model that attains the target metric. Once they first do, we assess their communication efficiency by examining the number of FL rounds they required ($\downarrow$). Table 13 illustrates this evaluation.

Cross-Silo. The cross-silo setting includes RoBERTa trained on the MRPC and RTE datasets, as outlined in Table 11.

⁴For each FEDOPT algorithm and dataset/task combination, we perform an exhaustive grid search to identify the best-performing hyper-parameters. For example, to optimize FEDADAM training RoBERTa on MRPC, we evaluate a 5×5 grid of client and server learning rate combinations—25 configurations in total—and select the one achieving the highest accuracy. Across all models, tasks, and optimizers, this process amounts to roughly 700 distinct training runs. While this substantial experimental effort is not emphasized in the main text due to space constraints, it underpins all reported results.

Table 13: Comparison of FEDOPT and FDA-OPT in communication-efficiency for training RoBERTa and DeBERTaV3 across different datasets to achieve target metrics. The table reports the number of rounds required to reach 85%, 90%, 95%, and 99% of the top-reported results in the literature (see Table 12)—naturally, the fewer rounds, the better ($\downarrow$). For example, in the bottom-left cell, FEDADAGRAD requires 29 rounds to achieve 90% of the best-reported accuracy for MRPC (i.e., at least $90\% \cdot 90.2 = 81.18\%$), while FDA-ADAGRAD achieves the same in 21 rounds, making it $1.4\times$ more efficient. Similarly, it has a speedup of $1.5\times$ for 95%.

	RoBERTa								DeBERTaV3			
	MRPC		RTE		SST-2		QNLI		MNLI-m		MNLI-mm	
	90% $\downarrow$	95% $\downarrow$	90% $\downarrow$	95% $\downarrow$	95% $\downarrow$	99% $\downarrow$	90% $\downarrow$	95% $\downarrow$	85% $\downarrow$	90% $\downarrow$	85% $\downarrow$	90% $\downarrow$
FEDAVG	26	30	67	103	8	93	35	71	12	16	12	15
FDA-SGD	6	10	16	20	7	43	9	23	4	8	4	6
FEDAVGM	21	31	111	182	30	91	38	81	29	35	29	35
FDA-SGDM	5	16	41	126	12	41	11	44	12	15	12	15
FEDADAM	20	40	118	153	12	42	11	22	11	12	11	12
FDA-ADAM	7	11	72	188	8	25	6	20	9	10	9	10
FEDADAMW	17	25	118	160	12	35	11	22	57	66	57	66
FDA-ADAMW	14	16	71	176	8	14	6	20	43	54	42	52
FEDADAGRAD	29	35	20	96	21	67	12	24	8	15	8	11
FDA-ADAGRAD	21	23	18	54	7	24	6	20	14	15	6	14
Avg. Speedup	$2.8\times$	$2.3\times$	$2.3\times$	$2\times$	$1.9\times$	$2.3\times$	$2.6\times$	$1.6\times$	$1.7\times$	$1.6\times$	$1.9\times$	$1.6\times$

- **RoBERTa on MRPC.** As per Table 13, FDA-OPT demonstrates superior communication-efficiency, with an average speedup of $2.3\times$ - $2.8\times$ compared to FEDOPT. It is $2.8\times$ more efficient in reaching 90% of the target metric, and $2.3\times$ for 95%.
- **RoBERTa on RTE.** Similar trends are observed for the RTE dataset. FDA-OPT achieves average speedups of $2\times$ - $2.3\times$ over FEDOPT, with specific gains of $2.3\times$ for 90%, and $2\times$ for 95% of the target metrics.

When the goal is to train the most accurate model—defined here as achieving 95% of the top-reported metrics—FDA-OPT is, on average, $2.15\times$ more communication-efficient in the cross-silo setting.

Cross-Device. In the cross-device setting, we analyze performance of training RoBERTa on the SST-2 and QNLI datasets, and DeBERTaV3 on MNLI-m and MNLI-mm, as outlined in Table 11.

- **RoBERTa on SST-2.** FDA-OPT consistently outperforms FEDOPT, achieving an improvement in communication-efficiency of $1.9\times$ - $2.3\times$. Specifically, FDA-OPT is $1.9\times$ more efficient in achieving 90% of the target metric, and $2.3\times$ for 99%.
- **RoBERTa on QNLI.** Again, FDA-OPT exhibits speedups ranging from $1.6\times$ - $2.6\times$. It is $2.6\times$ more efficient at 90%, and $1.6\times$ at 95% of the target metric.

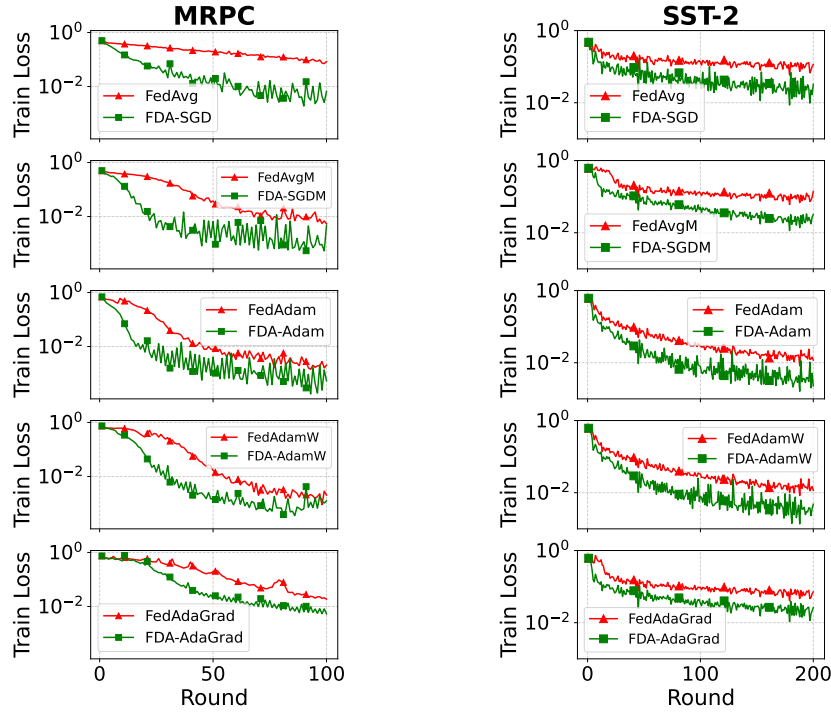


Figure 42: Training Loss progression of training RoBERTa with FDA-OPT vs. FEDOPT on MRPC (left) and SST-2 (right)

- **DeBERTaV3 on MNLI-m.** Similarly, FDA-OPT has improvements of $1.7\times$ and $1.6\times$, corresponding to 85% and 90% of the target metrics, respectively.
- **DeBERTaV3 on MNLI-mm.** Lastly, FDA-OPT is $1.7\times$ and $1.6\times$ more communication-efficient than FEDOPT, for attaining 85% and 90% of the target accuracies, respectively.

Similarly, in the cross-device setting, FDA-OPT achieves an average improvement in communication-efficiency of $1.8\times$ when training models to attain the highest possible accuracy targets.

Convergence. Analyzing training loss progression is key to understanding the convergence behavior of FL algorithms. It provides insights into how quickly and effectively models learn—essentially, their ability to minimize objectives. In this subsection we analyze Figure 42 which compares the performance of FDA-OPT and FEDOPT on two representative datasets from cross-silo and cross-device settings. The conclusions drawn regarding convergence pace and quality from these two datasets are indicative of the broader trends observed across all experiments.

Cross-Silo. On the MRPC dataset—which simulates a cross-silo setting following Table 11—all FDA-OPT algorithms demonstrate significantly faster convergence than their FEDOPT counterparts (Figure 42). For instance, FDA-SGD achieves a training loss $100\times$ smaller than FEDAVG within the first 100 rounds. Additionally, both FDA-SGDM and FDA-AdamW exhibit sharp drops within the initial 30–40 rounds, outperforming FEDAVGM and FEDADAMW, respectively, with FDA-SGDM stabilizing at a loss $10\times$ smaller than FEDAVGM. Meanwhile,

FDA-ADAM converges to a loss $5\times$ smaller than FEDADAM, and FDA-ADAGRAD consistently maintains a $5\times$ lower loss than FEDADAGRAD after the initial rounds.

Cross-Device. On the SST-2 dataset—representing a cross-device setting as outlined in Table 11—all FDA-OPT algorithms consistently achieve $5\text{--}10\times$ lower training loss compared to their FEDOPT counterparts (Figure 42). Unlike the cross-silo case, the initial drop in training loss is less pronounced, which is expected since each round involves only a small subset of clients. This leads to initial updates that are less representative of the overall training distribution.

4.4 Component Details - FDA-OPT

4.4.1 Standalone Architecture

Our internal component architecture has been designed to fully support the concurrent execution of multiple Federated Learning (FL) jobs, encompassing both traditional FL algorithms and our novel FDA method. Each job can arrive dynamically and is described by a dedicated configuration that specifies various parameters—such as the number of participating devices, the deep learning model to be used, the optimization strategy, and other runtime settings. Crucially, all FL jobs are completely isolated and independent from one another, ensuring that they do not interfere and can run in parallel or in succession without conflict.

To coordinate the lifecycle of these FL jobs and automate their deployment and management, we introduce a dedicated orchestration component referred to as the *mediator*. The mediator is responsible for supervising and handling every phase of a job's execution: this includes actions that occur *before the job starts, during its execution, and after the job completes*.

The mediator operates as a persistent entity that continuously listens to a specific Kafka topic. This Kafka topic serves as the communication channel over which new FL jobs are submitted to the system. These jobs are encoded as JSON messages that include all the necessary configuration details. Once the mediator receives such a JSON message from Kafka, it begins executing a series of steps to bring the job to life. Specifically, the mediator performs the following actions:

1. **Parsing the configuration:** The mediator first extracts and interprets the content of the JSON message. This configuration defines how the FL job should behave including the number of clients, the optimizer, networking details and more.
2. **Cluster setup:** Based on the parsed configuration, the mediator proceeds to initialize the Federated Learning environment. This involves launching a central FL server (via `server.py`) and spawning the specified number of FL clients (via `client.py`), each of which will train on its local dataset.
3. **Connecting components:** The mediator ensures that all launched components—the server and the clients—are properly connected to each other using the provided network addresses, ports, or other communication details included in the configuration.
4. **Verifying launch success:** Before considering the FL job as active, the mediator performs final checks to ensure that the entire deployed FL cluster has been booted up correctly and without errors. Only then does the training process begin.

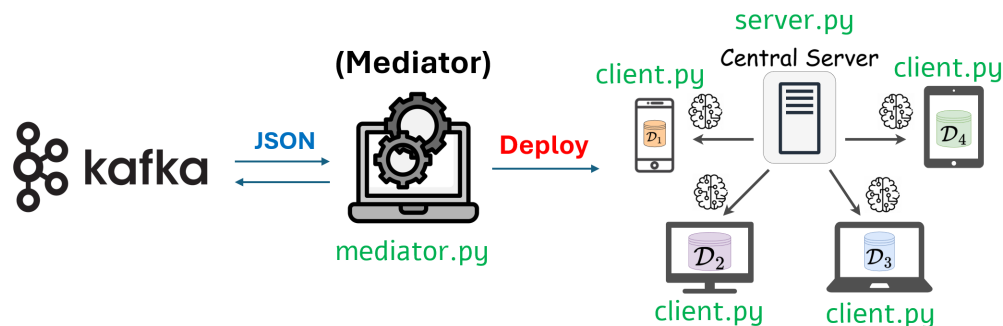


Figure 43: The *mediator* acts as a persistent orchestrator that continuously monitors a dedicated *kafka* topic for incoming FL job requests, which are transmitted in the form of structured JSON messages. Upon receiving such a job specification, the mediator proceeds to parse the configuration and automatically deploy a Federated Learning (FL) cluster tailored to the received settings

Figure 43 visually illustrates the aforementioned procedure. On the left, we see Kafka feeding job descriptions to the mediator in the form of JSON messages. The mediator, shown in the center, takes responsibility for deploying a fully functional FL setup on the right side of the figure, consisting of a central server and multiple clients (each with its own local dataset). Each client and the server run their respective scripts (`client.py` and `server.py`) as shown. This setup supports clean orchestration and job isolation across diverse FL configurations.

4.4.2 Configurations

Every federated learning (FL) job is fully specified by a JSON payload, which is pushed through Kafka and parsed by the mediator. This configuration controls every aspect of the FL cluster—from the dataset and model to the training schedule and network setup. The schema supports a wide range of combinations, allowing users to tailor jobs to their needs. Below, we show a minimal working example of a valid configuration, grouped by logical blocks such as `model`, `dataset`, `training`, `server`, and `clients`. Later sections will go over the details of each of those fields; the full JSON schema and explanation can be also be found at GitHub ⁵.

⁵<https://github.com/altairengineering/crexdata-public/tree/main/CREXDATA%20System%20Components/Federated%20Learning%20%28TUC%29/FDA-Opt-Sys>

Example JSON Configuration (truncated)

```

1  {
2      "model": {
3          "checkpoint": "roberta-base",
4          "num_labels": 2
5      },
6      "dataset": {
7          "name": "mrpc",
8          ...
9      },
10     "training": {
11         "num_clients": 10,
12         "clients_per_round": 2,
13         ...
14     },
15     "server": {
16         "network": {
17             "ip": "localhost",
18             "port": 8083,
19             ...
20         },
21         "strategy": {
22             "name": "FedAdam",
23             "fda": true,
24             ...
25         }
26     },
27     "clients": {
28         "network": [
29             {"id": 0, "ip": "localhost",
30              ↪ "port": 9199, "data_path": ""},
31             {"id": 1, "ip": "localhost",
32              ↪ "port": 9201, "data_path": ""},
33             ...
34         ]
35     }

```

4.4.3 Integration: With or Without Graphical Interface

One of the key strengths of our FL component is its modular and standalone design. We have integrated it as a functional unit within the broader CREXDATA ecosystem, allowing it to be triggered directly through the system's graphical interface, specifically via RapidMiner Studio. This means users can interact with it visually, making configuration and job submission accessible to non-technical stakeholders.

At the same time, the component is fully operable without the graphical interface. In headless deployments or automated pipelines, the system can be executed directly from the command line by simply running the `mediator.py` script. In both cases, the mediator behaves identically: it listens to a Kafka topic and reacts to incoming JSON job descriptions.

This flexibility ensures that our federated learning module is agnostic to the surrounding infrastructure and can be deployed in a wide range of environments—from fully integrated GUI-based workflows to lightweight, script-driven setups. This makes it especially useful for both production use and research prototyping.

4.5 NS3: Accurate Network Modeling

Real world Federated Learning (FL) environments are often characterized by *highly variable* and *unpredictable network behavior*. Typical distributed setups consist of multiple edge devices acting as clients, each connected through networks with different capacities and stability levels. Thus, variation in *bandwidth*, *client mobility*, and *channel reliability* can cause uneven training progress, distort synchronization timing, and ultimately affect convergence.

Despite the undeniable effect of network conditions in the overall performance and tuning of FL frameworks, many experimental studies are evaluated under *ideal communication*

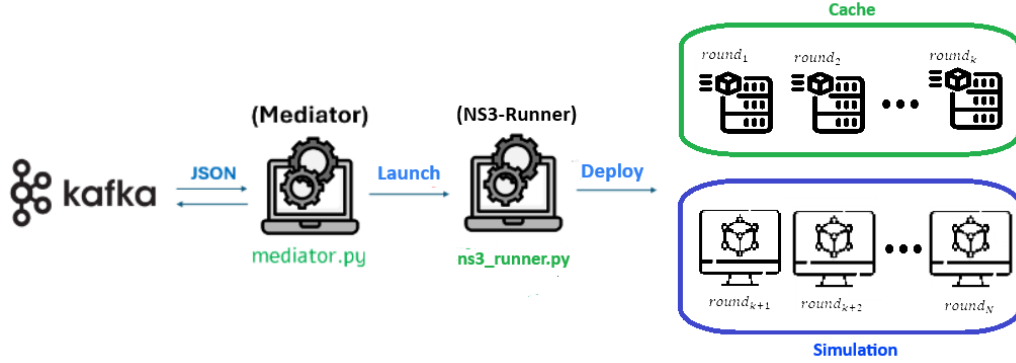


Figure 44: Upon receiving a simulation request, the *mediator* spawns an NS3-Runner to execute a single NS-3 job. Results are retrieved from cache when available, or generated through parallel round simulations. In the presented example we assume the first k rounds are cached while the rest are simulated.

assumptions overlooking the impact of network variability on the system behavior. This discrepancy between experimental and real-life conditions can lead to *poorly optimized frameworks* that fail to meet expected performance standards once deployed.

To better capture these dynamics under controlled yet realistic conditions, we integrate the discrete-event network simulator NS3 into the *CREXDATA* framework. Through this integration, we create a reproducible environment capable of modeling *heterogeneous topologies* and *traffic patterns*, enabling quantitative analysis of *communication overhead* and robustness evaluation to *unstable conditions*. This integration also enables systematic *hyperparameter tuning* and fine-grained control of *network presets* — a critical capability for analyzing how *latency variability* affects synchronization, convergence, and the balance between *communication and computation* under real-world conditions.

Integration Design The NS-3 integration extends the mediator-based orchestration framework described in Section 4.4.1 by introducing a modular *network simulation layer* as shown in Fig. 44. The mediator functions as a continuously running *coordination service* that exchanges simulation requests and results through *Kafka*. Each request includes basic topology information and selects one of three predefined *network presets*, which represent different performance levels characterized by unique combinations of *packet loss ratios*, *signal strengths*, and *communication protocols*. These presets encapsulate the main variability of network conditions while keeping configuration *accessible to non-expert users*.

Once a request is registered, the mediator launches a short-lived Python controller responsible for orchestrating a single simulation run. This component, referred to as the *ns3-runner*, determines the number of rounds specified in the request and initiates one NS-3 process for each simulated round. Parallelization is handled through Python’s *multiprocessing* module, as NS-3 supports only *single-process execution*. Each simulation instance is bound to a dedicated CPU core to ensure *full isolation* and to prevent interference with NS-3’s internal global state.

Efficient Tool Exploitation Upon completion, simulation results are serialized in *JSON format* and transmitted via *Kafka* while also being stored locally. If a subsequent request with identical configuration parameters is received, the *cached results* are retrieved immediately, avoiding redundant computation. This design significantly reduces processing time and enables efficient reuse of previously executed simulations. The result is a flexible and decoupled architecture that supports *concurrent, reproducible network evaluations* within the overall CREXDATA workflow. A sample output of the platform is provided for reference as follows:

Simulation Response JSON (Truncated)

```

1  {
2    "metadata": {
3      "config": {
4        "num_rounds": 8,
5        "num_cpus": 5,
6        "sync_method": "sync",
7        "monitor_msg_size": 0
8        "num_clients": 8,
9        "clients_per_round": 3,
10       "network_type": "wifi",
11       "server_datarate": "1000Mbps",
12       "tx_gain": 0.0,
13       "max_packet_size": 1500,
14       "network_standard": "basic",
15       "client_mobility": false,
16       "pid": "2a4c7bee93e2",
17       "model_size": 50.0,
18       "topology": {
19         "server": {"coordinates": [57.54, 58.62, 0]},
20         "clients": [
21           {"id": 0, "coordinates": [82.38, 82.67, 100]},
22           ... // Remaining clients truncated
23         ]
24       },
25     },
26   },
27   "rounds": [
28     {
29       "round": 1,
30       "records": [
31         {"id": 1, "connectionTime": 0.01, "downlinkTime": 27.0, "monitorigTime": 0.0,
↪      "uplinkTime": 16.53, "throughput": 24.2, "routerDistance": 33.89},
32         ... // remaining clients truncated
33       ]
34     },
35     ... // remaining rounds truncated
36   ]
37 }

```

The extracted simulation metrics can serve both as **diagnostic indicators** and as **actionable inputs** for system adaptation. From a *diagnostic perspective*, they enable the identification of clients that can reliably participate in communication, based on *connectivity quality* and *signal strength*, while also allowing quantitative estimation of the *computation-to-communication ratio*. This facilitates a direct comparison of synchronization strategies—such as *FDA-based* and

synchronous approaches—under varying network conditions, revealing how *communication degradation* accentuates the benefits of adaptive synchronization. In contrast, under *idealized, high-throughput networks*, these differences appear less pronounced, masking real-world performance dynamics.

From an *actionable standpoint*, the same metrics can drive *hyperparameter tuning* and *operational adjustments*. For instance, when simulations indicate slower or unstable connections, *larger synchronization thresholds* or *adjusted batch sizes* may be favored to reduce communication overhead. Conversely, in more stable network environments, *tighter synchronization intervals* and *lighter monitoring strategies* can improve responsiveness. This dynamic adaptation bridges network-level performance and learning behavior, ensuring that the federated process remains both *efficient* and *context-aware*.

Infrastructure Flexibility The seamless integration of the NS-3 tool on the standalone architecture presented in Section 4.4.1 further highlights the flexibility of our infrastructure. With the architecture of Figure 44, we extended the functionality of the mediator to handle dedicated network simulation requests. This design not only enables *automated and parallel execution of network experiments* but also preserves the same *dual execution pathways* available to the federated learning component—either through the graphical *RapidMiner* interface or as a *standalone process* executed directly via the `mediator.py` script. Consequently, users can configure and launch NS-3 simulations both through RapidMiner workflows and via local execution, using JSON-formatted configurations transmitted through the connected Kafka host. Both approaches follow the same schema and messaging workflow, ensuring *full interoperability* across execution modes.

4.6 An Adaptive NeRF Framework for City-Scale Emergency Awareness

Rapid situational awareness during emergencies such as *floods*, *earthquakes*, or *fires* is of paramount importance for effective response and mitigation. Traditional 3D mapping and visualization methods are inherently static, requiring full retraining whenever new data arrive—a process demanding extensive time and large curated datasets. In contrast, emergency data are streamed continuously from *aerial drones* in small, heterogeneous batches, making such static retraining infeasible. This limitation delays 3D scene updates and severely hinders first responders’ ability to *assess damage*, *locate survivors*, and *coordinate critical operations*.

To address this challenge, we developed a **city-scale neural radiance field (NeRF) framework** suitable for **high-speed adaptation and rendering under extreme data conditions**. Our system is first trained in the *offline stage* under normal conditions on aerial data from the target area, establishing strong priors that capture the scene’s *geometric* and *radiometric* structure. During an emergency, continuous streams of *drone-acquired data* are integrated into the NeRF, allowing it to update rapidly and reflect evolving conditions with minimal adaptation overhead.

Our approach leverages **meta-continual learning (MCL)** for fast adaptation without catastrophic forgetting, **modularized neural architectures** for spatial scalability, and **InstantNGP** for rapid optimization and compact representation. The combination of these components ultimately enables continuous, real-time refinement of the model as new data arrive, ensuring

up-to-date visual understanding during critical situations. This infrastructure bridges the gap between *high-fidelity city-scale 3D modeling* and *rapid adaptation* for emergency operations, demonstrating how *adaptive neural field methods* can transform situational awareness and decision-making in disaster management.

4.6.1 Foundational Methods and System Components

Meta-Continual Learning: Adaptive Initialization and Balanced Knowledge Integration

Our framework builds on **Meta-Continual Learning of Neural Fields (MCL-NF)** [90], which unifies **meta-learning** for rapid adaptation and **continual learning** for long-term stability. Meta-learning produces an initialization that embeds geometric and appearance priors of the environment, *enabling fast adaptation to new observations*. Continual learning allows these priors to evolve without being overwritten, *preserving structure while incorporating new content*. To prevent catastrophic forgetting, *MCL-NF* further introduces *sample-aware weighting* that *balances prior retention against novel information*. We leverage these properties to support stable online updates for streaming, heterogeneous drone observations while preserving spatial and photometric consistency at scale.

Spatial Decomposition: Modularized NeRF Architectures for Scalability City-scale representation demands scalability beyond the memory and training capacity of a single NeRF network. To scale to city-level scenes, we adopt *spatial decomposition* inspired by **Mega-NeRF** [80], splitting the environment into localized, independently optimized neural modules. At *runtime*, only modules intersecting new observations are updated, enabling targeted refinement without retraining the full scene. Cross-module continuity is maintained via *boundary-aware blending*, while the isolation of updates prevents cross-region interference, reducing memory, computation, and catastrophic forgetting risk.

InstantNGP: Rapid Optimization for Real-Time Adaptation We use **InstantNGP** [56] as the neural field backbone for each spatial module. Its *multiresolution hash encoding* enables compact representations, fast convergence, and tiny *MLP decoders*, making per-module updates lightweight enough for continuous *drone streaming*. The small memory footprint also minimizes model exchange cost in *distributed* or *bandwidth-limited* deployments, supporting near real-time assimilation of new imagery.

4.6.2 Pipeline Overview

Our system unifies these components into a scalable pipeline for *adaptive city-scale 3D reconstruction* within the CREXDATA ecosystem (Fig. 45). In the **Offline Stage**, modules are repartitioned spatially and meta-trained to acquire a shared initialization encoding the *geometric–radiometric priors* of the environment. In the **Runtime Stage**, new *drone imagery* is localized in the same coordinate frame and converted to rays that are *internally routed* to intersecting spatial experts. Modules are then refined via *lightweight gradient updates*, guided by *relevance-aware sample weighting* to integrate new content without degrading established structure. Combined with fast *InstantNGP* optimization and *modular isolation*, the pipeline delivers efficient, stable, and continually adaptive refinement of large-scale NeRF scenes under *streaming, time-critical* conditions.

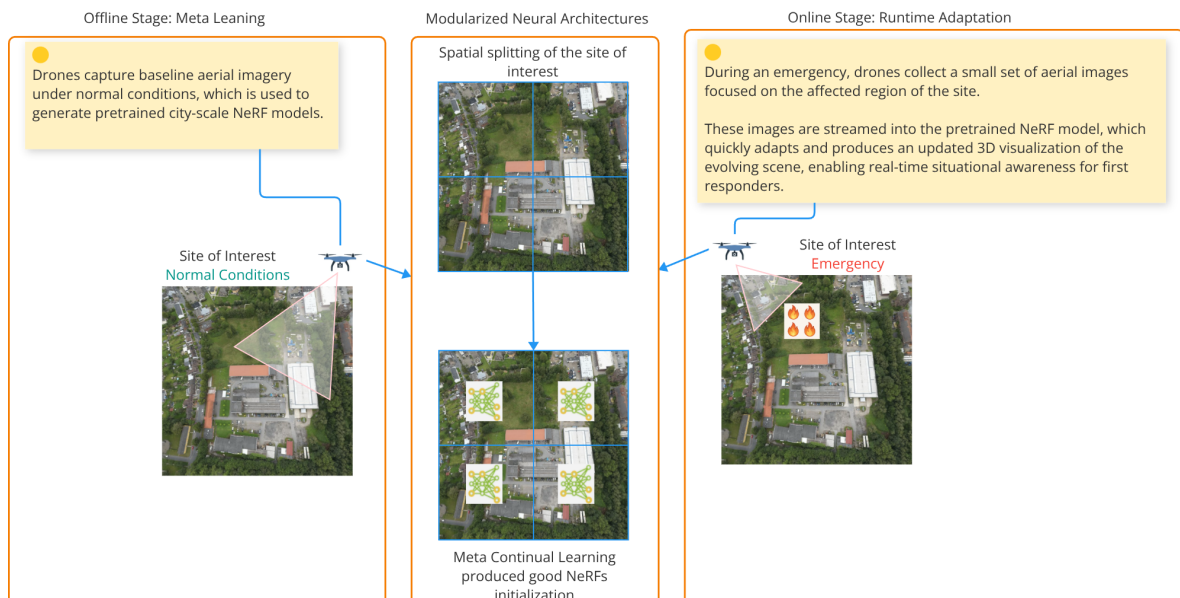


Figure 45: Overview of the proposed city-scale NeRF infrastructure. Drones collect baseline imagery under normal conditions to pretrain city-scale NeRF models. The environment is divided into spatial modules, each optimized via meta-continual learning for rapid adaptation. During emergencies, new drone imagery from affected areas is streamed into the corresponding modules, enabling fast NeRF updates and providing first responders with up-to-date 3D visualizations for situational awareness.

4.6.3 Offline Stage: Meta Learning

The first and most critical stage of our pipeline is performed *offline* proactively, under normal conditions. One or multiple drones are deployed over the area of interest to collect visual data that will be used for *meta-training*. The objective of this stage is to learn a scene-aware initialization that captures the *geometric* and *radiometric* structure of the environment, as illustrated in **Figure 46**. This process yields a coarse meta-learned initialization that encodes the dominant scene characteristics while avoiding overfitting to fine details.

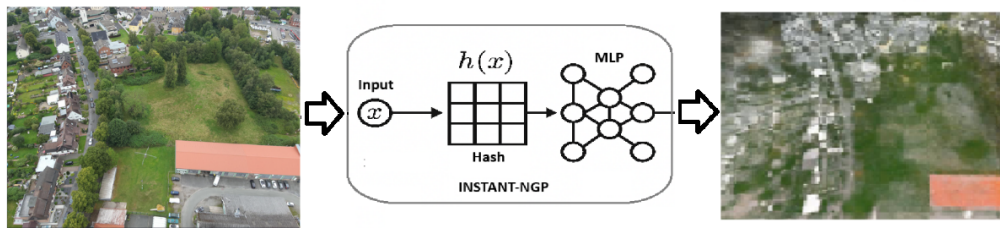


Figure 46: Example of meta-learned NeRF initialization for a single spatial region. The real scene (left) and the corresponding meta-learned representation (right) show that the model learns a coarse, structured prior rather than a fully detailed reconstruction

Data Preprocessing After imagery acquisition, a series of preprocessing steps are applied before data enters the meta-training pipeline, including **pose estimation via COLMAP**, **ray generation**, **spatial ray routing**, and **task construction**.

First the scene is reconstructed using **COLMAP Structure-from-Motion**, which estimates camera *intrinsic*s and *poses*. All poses are then expressed in a common **localized ENU (East–North–Up) coordinate frame** to ensure metric consistency across the reconstruction. If COLMAP outputs coordinates in **ECEF (Earth-Centered, Earth-Fixed)**, they are internally converted to ENU during preprocessing. This alignment anchors the reconstruction to a fixed reference point, providing a stable *origin*, *axes*, and *scale* that are shared by all cameras, 3D points, and subsequently generated rays.

For every pixel in each registered image, a **camera ray** is computed by projecting the pixel into world space using the aligned camera pose and intrinsic parameters, producing a *ray origin–direction pair* with an associated *ground-truth RGB value*. These ray samples form the fundamental training signal for all NeRF experts.

We adopt the **ray routing principle** introduced in *Mega-NeRF*, applying spatial partitioning so that each NeRF expert trains only on rays relevant to its own region. During preprocessing, we build a dedicated *regional ray dataset* for every expert, containing exclusively the rays routed to that region. Unlike hard single-region assignment, a ray is assigned to *every region it intersects*, enabling shared supervision near boundaries while preserving spatial locality. Rays that do not intersect a region are discarded early, leading to *smaller and more relevant training sets* with *lower memory and compute requirements*.

Task Definition Unlike typical NeRF approaches, which optimizes a single continuous scene using all available rays, meta-learning instead trains a model to solve many *small, independent reconstruction sub-problems*. Formally, a meta-learning task $T \sim p(T)$ defines *one such sub-problem*, structured as an adaptation episode sampled from a distribution of related tasks. Each task consists of two disjoint subsets drawn from a common task-specific distribution p_T : (i) a *support set* used for rapid model adaptation, and (ii) a *query set* used to evaluate the adapted model on previously unseen viewpoints within the same sub-problem.

In our formulation, the scene is spatially decomposed hierarchically. First, the area of interest is divided into a configurable number of **expert-owned regions**, a hyperparameter that determines modularization granularity. Each region is subdivided into a 3D grid with tunable resolution—shared across all regions for simplicity—where **each grid cell corresponds to one meta-learning task**. Each ray of the region is mapped to the cell that contains the largest portion of its path, ensuring it is assigned to the most relevant task. During training, cells are sampled randomly so that each expert learns to adapt across its entire region. For every sampled cell, its rays are split into support and query subsets with enforced *viewpoint and pixel disjointness*, training experts to produce initializations that can quickly adapt to any new cell from only a few previously unseen rays.

With offline data preparation in place, we ensure that rays are normalized into the expected NeRF input format, routed to the appropriate spatial expert based on their location, and organized into tasks. The full processing procedure of those tasks is summarized in Algorithm 2.

Algorithm 2 Meta-Continual Pipeline with Modularization

Require: $p(\mathcal{T})$ Distribution over tasks
Require: Spatial Regions $\mathcal{C} = \{1, \dots, C\}$ and $\forall c \in \mathcal{C}$: Region tasks $\mathcal{T}_c$, model parameters θ_c and using them, expert neural fields $f_c(\mathbf{x}) \equiv f(\mathbf{x}; \theta_c)$
Require: Hyperparameters: batch sizes $|S|$ and $|Q|$, learning rates n_{inner} and n_{outer} , inner loop iterations K

Offline Stage: Meta-Learning

```

1: while not done do ▷ each iteration processes all regions
2:   for each region  $c \in \mathcal{C}$  do
3:     Sample task batch  $\mathcal{B}_c \sim p(\mathcal{T}_c)$ 
4:     for each task  $T_i \in \mathcal{B}_c$  do
5:       Sample support  $S_i$  and query  $Q_i$  sets
6:       Initialize Fisher Information Matrix  $F_i$  for task  $T_i$ 
7:       Inner loop (Task-specific Adaptation)
8:       for  $k = 1 \dots K$  do
9:         Compute predictions:  $\hat{\mathbf{y}} = f_c^{(k-1)}(\mathbf{x}), \forall (\mathbf{x}, \mathbf{y}) \in S_i$ 
10:        Compute FIM weight  $w_{\text{fim}}$ .
11:        Computed FIM-weighted loss:  $\mathcal{L}_{\text{inner}}^{c,i} \leftarrow \frac{1}{|S|} \sum_{(\mathbf{x}, \mathbf{y}) \in S_i} (w_{\text{fim}} \cdot \|\hat{\mathbf{y}} - \mathbf{y}\|^2)$ 
12:        Update fast parameters:  $\theta_c^{(k)} \leftarrow \theta_c^{(k-1)} - n_{\text{inner}} \nabla_{\theta_c^{(k-1)}} \mathcal{L}_{\text{inner}}^{c,i}$ 
13:        Update Fisher Information Matrix  $F_i$ .
14:       end for
15:       Query Evaluation
16:       Compute task  $i$  loss:  $\mathcal{L}_{\text{meta}}^{c,i} \leftarrow \frac{1}{|Q|} \sum_{(\mathbf{x}, \mathbf{y}) \in Q_i} (w_{\text{fim}} \cdot \|\hat{\mathbf{y}} - \mathbf{y}\|^2)$  as in steps 9-11
17:     end for
18:   end for
19:   Meta-update
20:   Update expert initialization:  $\theta'_c \leftarrow \theta_c - n_{\text{outer}} \nabla_{\theta_c} \left( \frac{1}{|\mathcal{B}_c|} \sum_{T_i \in \mathcal{B}_c} \mathcal{L}_{\text{meta}}^{c,i} \right) \quad \forall c \in \mathcal{C}$ 
21: end while

Online Stage: Runtime Adaptation
22: for each incoming view batch  $I$  do
23:   Ray generation:  $R \leftarrow \text{GenerateRays}(I)$ 
24:   for  $k = 1 \dots K$  do ▷ as in steps 9–13, ray-based
25:     Sample support rays  $S \subseteq R$ 
26:     Compute routing weights  $G_c(S) \quad \forall c \in \mathcal{C}$ 
27:     Compute expert predictions  $\hat{\mathbf{y}}_c \leftarrow f_c(S) \quad \forall c \in \mathcal{C}$ 
28:     Blend output  $\hat{\mathbf{y}} \leftarrow \sum_{c=1}^C G_c(S) \cdot \hat{\mathbf{y}}_c$ 
29:     Compute loss;  $L \leftarrow \frac{1}{|S|} \sum_{(\mathbf{x}, \mathbf{y}) \in S} \|\hat{\mathbf{y}}(\mathbf{x}) - \mathbf{y}\|^2$ 
30:     Update experts:  $\theta'_c \leftarrow \theta_c - n_{\text{inner}} \nabla_{\theta_c} L \quad \forall c \in \{c \in \mathcal{C} \mid G_c(S) > 0\}$ 
31:   end for
32: end for

```

Meta Training Process Each spatial region $c \in \mathcal{C}$ with task distribution $p(\mathcal{T}_c)$, where $\mathcal{C}$ is the set of all regions, is assigned to a *regional expert* with initial model parameters θ_c . The region's expert is a *neural field* f_c mapping input vector $\mathbf{x}$ to radiance estimates $\hat{\mathbf{y}}$, with $\mathbf{y}$ being the ground truth value⁶. During meta-learning, batches of tasks $T_i \sim p(\mathcal{T}_c)$, denoted as $\mathcal{B}_c$, are repeatedly sampled for each region c . Each task T_i is composed of a small *support set* S_i and a disjoint *query set* Q_i with fixed $|S|$ and $|Q|$ rays respectively per task.

⁶A neural field is a continuous function $f_\theta : \mathbb{R}^n \rightarrow \mathbb{R}^m$ parametrized by an MLP, mapping coordinates to signals. In NeRF, $f_\theta(\mathbf{x}, d) \rightarrow (\sigma, c)$, predicting density and RGB color. For simplicity, we denote both input and output compactly as vectors $\mathbf{x}$ and $\mathbf{y}$.

For every task T_i , the expert of region c starts with the initialization $\theta_c^{(0)}$ and undergoes K adaptation steps on the support set S_i , minimizing the Fisher-weighted loss at each step k :

$$\mathcal{L}_{\text{inner}}^{c,i} = \frac{1}{|S|} \sum_{(x,y) \in S_i} w_{\text{fim}} \cdot \|f_c^{(k-1)}(x) - y\|^2,$$

$w_{\text{fim}}(\cdot)$ is derived from the diagonal of the **Fisher Information Matrix (FIM)**, which estimates how sensitive the model output is to each parameter. In practice, *FIM* acts as a memory-awareness signal: parameters that strongly influence previously learned structure receive smaller updates, while less critical parameters remain free to change. This keeps adaptation *plastic*—able to learn new local details quickly—but also *non-destructive*, preventing abrupt shifts in parameters that encode stable scene properties [90].

During the **inner loop**, each expert is trained to rapidly tune itself to a new localized radiance field configuration (e.g., transient geometric changes, new occlusions, local illumination shifts) using only a small set of incoming rays, mimicking the sparse and incremental nature of drone observations at test time.

After each adaptation step k , the expert’s temporary parameters are written as **fast weights** $\theta_c^{(k)}$, and are updated with learning rate n_{inner} as follows:

$$\theta_c^{(k)} = \theta_c^{(k-1)} - n_{\text{inner}} \nabla_{\theta_c^{(k-1)}} \mathcal{L}_{\text{inner}}^{c,i}$$

The fast parameters of the last iteration $\theta_c^{(K)}$ are propagated for the task’s **Query Evaluation**, where the query loss –meta-update signal– is computed over samples from the query set Q_i :

$$\mathcal{L}_{\text{meta}}^{c,i} = \frac{1}{|Q|} \sum_{(x,y) \in Q_i} w_{\text{fim}} \cdot \|f_c^{(K)}(x) - y\|^2.$$

This loss evaluates how well the expert can *generalize after adapting*, measuring its ability to render unseen rays from the same spatial cell derived from Q_i after being trained only on S_i . In essence, the goal is not to become perfect at fitting the support set, but to emerge from a few gradient steps with a model that predicts the rest of the local scene correctly. The principal idea is that *a good initialization is not one that already knows the answer, but one that learns it quickly from limited evidence*. Thus, only the *initialization* parameters θ_c of each region are optimized with respect to $\mathcal{L}_{\text{meta}}^{c,i}$ for all tasks $T_i \in B_c$ with learning rate n_{outer} :

$$\theta'_c = \theta_c - \eta_{\text{outer}} \nabla_{\theta_c} \left(\frac{1}{|B_c|} \sum_{T_i \in B_c} \mathcal{L}_{\text{meta}}^{c,i} \right) \quad \forall c \in \mathcal{C}$$

To summarize the meta-learning procedure is structured around two synergistic optimization levels with a deliberate distinction:

- The **inner loop - Task Adaptation** performs *fast local learning*, adjusting the model to fit a specific spatial cell using only its support rays.

- The **Query evaluation** performs *learning-to-adapt*, shaping the initial parameters so that such small local updates consistently lead to strong generalization on unseen rays from the same region.

Continual-Learning While the *meta-learning* concept ensures rapid adaptation, the **continual learning** aspect is achieved through two complementary design properties. First, the spatial modularization itself enforces inherent locality: each expert operates on its own spatial region, meaning that gradient updates caused by new observations only affect a single module and do not propagate to others, naturally preventing cross-region interference. Second, within each region, the *Fisher-weighted* inner updates explicitly guard previously acquired knowledge by attenuating gradient magnitudes for parameters associated with stable scene structure, thereby mitigating catastrophic forgetting within a single region. Together, modular isolation and Fisher-aware adaptation provide safeguards at both the *inter-region* and *intra-region* levels, enabling continual learning without degrading prior knowledge.

Large-scale Optimizations Scaling *meta-continual learning* to large 3D scenes introduces strict computational and memory constraints, which we address through two design choices. First, each spatial expert is built on **Instant-NGP**, whose multiresolution hash encoding enables compact yet expressive representations while keeping the MLP decoders extremely small [56], making NeRF experts lightweight and efficient—essential when maintaining many regional models. Second, while **MAML** [22] enables rapid task adaptation, its exact formulation is prohibitively expensive at scale, as the meta-gradient must backpropagate through the full inner-loop optimization trajectory $\theta_c^{(0)} \rightarrow \theta_c^{(1)} \rightarrow \dots \rightarrow \theta_c^{(K)}$, requiring second-order derivatives and large memory to retain intermediate states. To remove this bottleneck, we adopt **First-Order MAML (FO-MAML)**, which approximates the meta-gradient by detaching the inner-loop trajectory, treating the adapted parameters $\theta_c^{(K)}$ as constants during the outer update and bypassing all second-order terms, yielding the following approximation:

$$\nabla_{\theta_c} \mathcal{L}_{\text{meta}}^c \approx \nabla_{\theta_c^{(K)}} \mathcal{L}_{\text{meta}}^c$$

Combined, these two design choices—*compact Instant-NGP experts* and *lightweight FO-MAML optimization*—make large-scale meta-training computationally practical, while sustaining high-quality results..

Altogether, by combining **meta-continual learning**, **region-wise modularization**, and efficient optimization of **Instant-NGP experts**, the system forms a scalable NeRF framework that delivers *fast local adaptation to newly observed data* while preserving *global stability* and *reconstruction fidelity* throughout the scene.

4.6.4 Online Stage: Runtime Adaptation

The **Online Stage of Runtime Adaptation** performs a lightweight, deployment-time specialization of the meta-learned model. Unlike the offline stage, this phase does not construct tasks, partition datasets, or optimize a meta-objective. Instead, the system operates as a

⁷This relationship is used to enhance efficiency on step 20 of Algorithm 2

unified NeRF model, in which all experts function as components of a single shared NeRF, queried jointly over the full spatial domain.

At deployment time, novel drone imagery arrives in batches I , representing the most recent observations of the environment. Each incoming image is processed through the same COLMAP pipeline used during the offline stage, estimating camera intrinsics and poses. Rather than reconstructing a new scene, we *append* these poses and images to the existing SfM reconstruction, preserving the global coordinate frame. This guarantees metric consistency across time and ensures that newly streamed observations are immediately routable to the correct spatial expert, without altering previously learned geometry. Ray generation then proceeds as in **Section 4.6.3**, producing the adaptation ray set R .

Routing is handled internally by the container NeRF. To remain consistent between meta-training and deployment, each ray is forwarded to all experts whose spatial region intersects the ray. Regions are optionally expanded by a boundary margin m , allowing rays near borders to be shared across neighboring experts, which reduces edge discontinuities with minimal computational overhead. In practice, m is a small fraction of the region size (e.g., 5 – 15%)⁸, enabling two routing modes based on its value:

- **Hard Routing** ($m = 1$): Each ray is assigned to exactly one expert—the one whose centroid $\mathbf{c}_k$ is closest to the ray sample position $\mathbf{x}$:

$$k^* = \arg \min_k \|\mathbf{x} - \mathbf{c}_k\|_2$$

Only that expert contributes to the output.

- **Soft Routing** ($m > 1$): A spatial overlap region is enabled, allowing multiple experts to contribute. Each contributing expert receives an inverse-distance score

$$\begin{cases} \tilde{w}_k = \frac{1}{\|\mathbf{x} - \mathbf{c}_k\|_2 + \varepsilon} \\ w_k = \frac{\tilde{w}_k}{\sum_j \tilde{w}_j} \end{cases} \quad \text{with} \quad \sum_k w_k = 1$$

The container **NeRF prediction** becomes a weighted blend of expert outputs:

$$\mathbf{f}(\mathbf{x}) = \sum_k w_k \mathbf{f}_k(\mathbf{x}) \tag{10}$$

At runtime we adapt the container NeRF for K iterations. In each iteration we sample a small support set of rays $S \subset R$ and compute routing weights $G_c(S)$ that determine which experts contribute to each ray. The routing rule implements either *hard routing* (one-hot assignment to the nearest expert) or *soft routing* (spatial overlap allowed, weights derived from inverse distance). The blended prediction for a ray is produced as in **Eq. 10** and we minimize the standard reconstruction loss – not the FIM weighted loss, as there is no meta-continual signal.

$$L = \frac{1}{|S|} \sum_{(x,y) \in S} \|\hat{\mathbf{g}}(x) - \mathbf{y}\|^2$$

⁸At runtime, the boundary margin must satisfy $m_{\text{runtime}} \leq m_{\text{train}}$, otherwise rays may be routed to experts that were never trained on the overlapping parts of the regions.

Only the experts with non-zero routing weight are updated:

$$\theta'_c = \theta_c - n_{inner} \nabla_{\theta_c} L \quad \forall c \in \{c \in \mathcal{C} \mid G_c(S) > 0\}$$

In other words, during deployment we only perform the **Inner-Loop (Task Adaptation)** stage, as we no longer try to meta-learn a good initialization, but instead try to *overfit the incoming data* to produce high-fidelity outputs.

4.6.5 Rendering and Use Case Integration

To visualize our learned neural field we integrate a lightweight interactive renderer built on *nerfview* and the *viser* backend, exposing a real-time 3D viewer with dynamic camera control.

NeRF renderers operate by sampling rays from a virtual camera, querying a neural field at many 3D points along each ray, and compositing the returned colors and densities into pixels. When the underlying field is well-trained, the renderer can synthesize views from arbitrary poses, effectively turning the model into a continuous, navigable 3D scene. In our framework, we plug in the learned multi-expert field by replacing the default network query with our *spatial routing* and *expert blending* module, allowing the viewer to render directly from our model while preserving standard ray sampling and image formation. This linkage enables real-time exploration of the reconstructed space with efficient view synthesis via *volumetric rendering*.

Operation Modalities The viewer exposes two modes: (i) **View-Model**, enabling free navigation and inspection of the reconstructed city-scale neural field, and (ii) **Runtime-Adaptation**, which visualizes streaming updates including ray routing, expert activation, and parameter evolution during online learning (**Sec. 4.6.4**). Users can seamlessly toggle between exploring a static model or observing a live adapting one.



Figure 47: Viewer instance demonstrating the interactive control API. On the left we have the rendered view and the right the controls for seamless user interaction.

Interactive Controls A graphical control panel enables real-time adjustment of rendering and learning parameters as shown in **Fig. 47**. Rendering options include resolution and frame-rate targets, output buffers (RGB, depth, and density accumulation with selectable color-maps), and near–far range clipping to constrain ray sampling bounds. Navigation utilities support instant re-centering, horizon alignment, top-down view, and pose resets for rapid scene traversal. Additionally, users can isolate individual spatial experts by visualizing the output of a single submodule, or switch to the blended multi-expert rendering to validate integration behavior. During runtime adaptation, additional controls unlock learning configuration, such as step count, learning rates, batch sizes, checkpoint export and GPU utilization controls like active rays per batch and discrete 3D samples processed per ray.

Adaptation Lifecycle and Checkpoints To initiate live adaptation, users provide a directory of newly captured imagery; once verified, the interface transitions into training mode with full optimization and monitoring support. While runtime-adapted expert checkpoints can be exported for inspection or short-term reuse, they should *not be handled as initializations for new scenes*. Runtime adaptation overfits to a specific distribution and erodes general priors, therefore users could benefit from the generalized meta-initialization on new environments instead of continuing adaption upon prior specialized instances. This further highlights the flexibility of our method, as one meta-learned initialization can be employed for multiple critical-condition future scenarios.

Project Integration Our NeRF pipeline fits seamlessly into the CREXDATA architecture, providing flexible execution modes for both expert and non-technical users. Specifically, all stages of the pipeline—offline meta-training and online adaptation—can be launched using lightweight configuration inputs that enable parametrized execution within the CREXDATA ecosystem. When used within RapidMiner, the pipeline appears as a graphical operator whose parameters are configured in the workflow interface and transmitted as a simple JSON file, enabling seamless integration with upstream data providers and downstream analytic handlers. The same processes can be executed from Python scripts using command-line arguments that map directly onto the corresponding configuration fields, offering a script-driven alternative suitable for automated pipelines and rapid prototyping in environments lacking graphical interface support. While the two entry points differ in how parameters are supplied, they follow the same execution logic and map to the same underlying configuration structure, ensuring consistent behavior across GUI-based and script-driven integration contexts.

4.7 Uncertainty-aware Synchronization Protocols

In many federated learning (FL) deployments (IoT, wearables, robots, industrial sensing), training data are collected through measurement pipelines that are inherently noisy due to sensor noise, calibration error, quantization, and clock drift. Let the clean (latent) input be x , while the client observes $\tilde{x} = x + \eta$, where η captures these perturbations. Since learned parameters depend on the observed data, we model the client parameters at step t as a random vector $W_k^{(t)}$ representing plausible model instances consistent with noisy observations.

A convenient approximation is Gaussian uncertainty, i.e., $W_k^{(t)} \sim \mathcal{N}(\mu_k^{(t)}, \Sigma_k^{(t)})$. This is motivated by (i) approximately Gaussian sensing/calibration errors and (ii) local smoothness of the

data-to-parameters mapping for small perturbations. Linearizing around the nominal dataset yields $W_k^{(t)} \approx w_k^{(t)} + J_k^{(t)} \xi_k$ and $\Sigma_k^{(t)} \approx J_k^{(t)} \text{Cov}(\xi_k) (J_k^{(t)})^\top$, where ξ_k stacks measurement/calibration errors and $J_k^{(t)}$ is the sensitivity of the training outcome to these perturbations. If ξ_k is Gaussian then the affine transform is Gaussian. Otherwise it still provides a tractable surrogate specified by mean and covariance.

Given K clients, we define the random global average $\bar{W}^{(t)} = \frac{1}{K} \sum_{k=1}^K W_k^{(t)}$ and measure cross-client divergence via the model variance $\text{Var}(W^{(t)}) = \frac{1}{K} \sum_{k=1}^K \|W_k^{(t)} - \bar{W}^{(t)}\|_2^2$. We aim to continue local training without synchronizing only when the divergence is below a threshold Θ with confidence at least $1 - \alpha$, i.e., $\Pr(\text{Var}(W^{(t)}) \leq \Theta) \geq 1 - \alpha$.

Direct evaluation of this equation is expensive because we would need to compute the covariance matrix for neural networks with thousands or millions of weights. So we resort to an upper $(1 - \alpha)$ confidence bound (UCB) $\tilde{B}^{(t)}$ such that $\Pr(\text{Var}(W^{(t)}) \leq \tilde{B}^{(t)}) \geq 1 - \alpha$, and we trigger synchronization using the sufficient condition $\tilde{B}^{(t)} > \Theta$. If we write $W_k^{(t)} = \mu_k^{(t)} + \varepsilon_k^{(t)}$ with $\varepsilon_k^{(t)} \sim \mathcal{N}(0, \Sigma_k^{(t)})$. Then, the Mahalanobis norm $(\varepsilon_k^{(t)})^\top (\Sigma_k^{(t)})^{-1} \varepsilon_k^{(t)} \sim \chi_p^2$, where χ_p^2 is the chi-square distribution with p degrees of freedom. Let $q_{1-\alpha/K}$ be the $(1 - \alpha/K)$ -quantile of χ_p^2 , implying $\Pr[(\varepsilon_k^{(t)})^\top (\Sigma_k^{(t)})^{-1} \varepsilon_k^{(t)} \leq q_{1-\alpha/K}] = 1 - \alpha/K$. Using $(\varepsilon^\top \Sigma^{-1} \varepsilon) \geq \|\varepsilon\|_2^2 / \lambda_{\max}(\Sigma)$ yields the Euclidean bound $\Pr[\|\varepsilon_k^{(t)}\|_2^2 \leq \lambda_{\max}(\Sigma_k^{(t)}) q_{1-\alpha/K}] \geq 1 - \alpha/K$, which justifies using $q_{1-\alpha/K}$ as a concentration factor in the trigger.

We treat the reported local model $w_k^{(t)}$ as the point estimate (mean proxy) $\mu_k^{(t)}$, i.e., $\mu_k^{(t)} := w_k^{(t)}$. We adopt the following conservative, aggregation-friendly bound: $B^{(t)} = \frac{1}{K} \sum_{k=1}^K \left(\|\mu_k^{(t)} - \bar{w}^{(t_{\text{sync}})}\|_2^2 + q_{1-\alpha/K} \text{tr}(\Sigma_k^{(t)}) \right)$, where we use $\lambda_{\max}(\Sigma_k^{(t)}) \leq \text{tr}(\Sigma_k^{(t)})$, and set $\tilde{B}^{(t)} := 2B^{(t)}$. The first term grows when client means drift apart; the second grows when measurement/calibration uncertainty is large, providing a conservative divergence estimate. To avoid transferring full covariances, we approximate $\Sigma_k^{(t)}$ as diagonal and transmit only its trace. In turn, the trace can also be effectively approximated by $\text{tr}(\Sigma_k^{(t)}) \approx c \cdot lr \cdot \sum_j v_{k,j}^{(t)}$, with lr being the current learning rate and $v_{k,j}$ being the gradient variance of client's k j^{th} parameter, anyway computed by optimizers such as Adam. Empirically, setting $c = 10^{2-\log_{10}(\#weights)}$ in our experiments.

Let $\bar{w}^{(t_{\text{sync}})}$ be the last synchronized global model. Each client sends two scalars, $\delta_k^{(t)} = \|w_k^{(t)} - \bar{w}^{(t_{\text{sync}})}\|_2^2$ and $s_k^{(t)} = c \cdot lr \cdot \sum_j v_{k,j}^{(t)}$, and the server computes $B^{(t)} = \frac{1}{K} \sum_{k=1}^K (\delta_k^{(t)} + q_{1-\alpha/K} s_k^{(t)})$. Set $\tilde{B}^{(t)} := 2B^{(t)}$. Synchronization is triggered iff $\tilde{B}^{(t)} > \Theta$. Otherwise, local training continues.

4.8 Lessons Learned

Employing federated learning techniques is a process that can be made accessible to non-expert users by using our auto-tuned techniques developed in this deliverable, but their effectiveness can be drastically improved (by orders of magnitude) through careful tuning. This was evident by the impressive improvements that we demonstrated in Deliverable D4.1 using a tuned Θ parameter as the threshold for requiring communication between distributed

workers. We also learned that a single optimization is not enough for near real-time NeRF computation for emergency scenarios, and we require the combination of federated learning techniques, combined with techniques for segmenting the space to create smaller models per area, combined with meta learning techniques and encoding optimizations.

5 Optimized Distributed Analytics as a Service

5.1 Overview of Optimization Approaches in CREXDATA

In Deliverable D4.1, we presented a suite of algorithms which are capable of leveraging parallelization in order to ensure rapid exploration of the search space. To achieve that, the algorithms were accompanied by a novel plan enumeration scheme and the basic concept was to model the search space as a Graph of Plans (GoP), where each vertex includes an entire, candidate physical workflow. The algorithms explore the search space using a number of parallel workers and by making the transition from one plan to another applying *actions*. Actions are changes of placements of operators on devices or execution platforms from the currently deployed plan, termed the *root plan*. Each such action is an edge in the GoP. This suite of algorithms, now termed APEROL, has been recently published in Banelas et al [70].

The APEROL algorithms presented in Deliverable D4.1 include an Exhaustive Search with Counting (ESC), a Random Sampling Search (RSS) and a Greedy Search with Progressive global optima (Greedy). In the current deliverable, in addition to the aforementioned APEROL algorithms, we introduce an Exhaustive Search with a Queue (ESQ) algorithm and a Heuristic search algorithm (Heuristic). The newly introduced algorithms abide by the concept of exploring the GoP using parallel workers, each examining the cost of a candidate physical plan and how much it improves the root plan (if at all).

Despite the rapid exploration of the search space induced by the APEROL suite [70] of parallel optimization algorithms, none of the non-exhaustive algorithms guarantees to output an optimal plan. On the other hand, the exhaustive APEROL algorithms will never conclude upon exploring candidate plans in network settings composed by thousands of devices. Therefore, we here also introduce an alternative modelling of the search space of candidate plans and a new (non-parallel) algorithm, namely DAG* [74]. For efficient exploration of the new search space of candidate physical workflows, DAG* relies on pruning huge amounts of the search space simultaneously guaranteeing optimality of the returned solutions. An important distinction is that APEROL can operate over a variety of optimization objective, while DAG* [74] is restricted to end-to-end latency of physical workflows.

Based on the above, in this chapter, we first complement our discussion of Deliverable D4.1 with the ESQ and Heuristic and their incorporation in the APEROL suite. We then proceed with elaborating on DAG*. We finally provide an experimental comparison of (i) APEROL suite of algorithms among themselves (ii) APEROL, DAG* vs state-of-the-art optimization algorithms that have been proposed in the literature.

5.2 Recap on APEROL's Problem Setup

CREXDATA assumes a *cloud-edge continuum* that encompasses everything from constrained sensors and mobile devices, through fog nodes, up to powerful cloud servers. Executing all extreme-scale analytics purely in the cloud is considered fundamentally non-scalable for the reasons explained in Deliverable D4.1. The preferred approach is therefore to *distribute* the operators of an extreme-scale analytics workflow across the network, exploiting device capacities and minimizing communication, latency, and energy usage. However, this turns the problem into a *combinatorial optimization* challenge: many interconnected operators, each

with multiple execution options and many candidate devices, interact in complex ways so that the placement of one operator affects the performance of others upstream/downstream in the workflow. Because in realistic scenarios there may be many workflows and many devices, manual placement is infeasible and error-prone. APEROL thus introduces an algorithmic optimization suite that automatically maps logical workflows to physical plans.

5.2.1 Logical Workflows vs. Physical Plans

A *logical workflow* is a graph of operators expressing the analytics semantics (filters, joins, aggregations, complex event operators, machine/deep learning components, user-defined functions, etc.) without execution details.

The optimizer's output is a *physical execution plan* (or *physical workflow*) that augments the logical graph with concrete execution decisions on certain devices and platforms for every operator. The novel enumeration scheme introduced in APEROL says: If there are O operators, P possible execution configurations/platforms per operator, and S sites, then the total number of possible physical plans is $(P \cdot S)^O$ which quickly becomes enormous even for modest O , P , and S .

APEROL formulates a *multi-objective* optimization problem. Each physical plan is evaluated on the following main performance dimensions:

- **T : Throughput** – number of input tuples processed per unit time.
- **L : Latency** – end-to-end delay, including both processing and network latency, from ingestion to output.
- **C : Communication cost** – total volume of data sent over the network (a proxy for bandwidth usage).
- **M : Migration Cost** – end-to-end delay, for transferring state of stateful operations upon operator re-assignment as a result of an new physical workflow (i.e., physical workflow adaptation decision).

The goal is to find physical plans that are best with respect to $[T, L, C, M]$. APEROL can select a physical plan using a convex combination (weighted sum) of the metrics, according to application priorities (e.g., low latency vs. low communication). The optimization is not a one-shot process. The underlying infrastructure and workload are dynamic as (i) New devices may join or leave the network, (ii) data distributions and workload intensity may change over time. A plan that was good at time t_0 may become suboptimal later. Hence, APEROL (i) Starts with an instantly devised, “good-enough” *root plan*, (ii) continues to run an optimization algorithm, exploring alternative physical plans, (iii) can switch to a better plan when the expected gains outweigh the migration cost.

Graph of Plans and Actions To structure the search space of possible execution plans, APEROL uses what was originally called a *Graph of Plans* (GOP). In the GoP, each vertex corresponds to a *physical workflow*. Conceptually, the vertex encapsulates a nested graph representing a fully specified physical workflow for the given logical workflow. All placement and execution decisions for all operators are fixed at a node. An edge between two vertices

in the graph of plans corresponds to applying a single *action* to one physical plan, yielding another. More precisely:

- An **edge** from plan *A* to plan *B* represents a single, atomic change in the configuration of exactly one operator (or, equivalently, one operator–site association).
- Applying one action produces a *1-hop neighbor* in the graph of plans; applying two actions produces a *2-hop neighbour*, and so on.

Formally, an *action* is a small, atomic modification to the configuration (site or platform change) of a *single* operator. In enumeration-based views (e.g., exhaustive and random sampling), these actions can be encoded as combinations of the form *option | site* for example $[P_1 | S_1, P_1 | S_2, P_2 | S_1, P_2 | S_2]$, where each operator chooses exactly one such combination. Each action also has an associated *migration cost*. During the transition from one plan to another, there may be increased latency, temporary zero throughput, and state-transfer overhead. A candidate plan is therefore attractive only if the performance improvement compensates for the migration overhead. Denoting a generic cost function by $\text{Cost}(\cdot)$ and the migration cost from the root plan to a candidate plan by $\text{MigCost}(\cdot)$, a simple decision criterion is:

$$\text{Cost}(\text{candidate}) + \text{MigCost}(\text{root} \rightarrow \text{candidate}) \ll \text{Cost}(\text{root}).$$

5.2.2 Algorithmic Exploration of the Graph of Plans

The APEROL optimization suite as presented in Deliverable D4.1 included (ESC, Greedy, RSS) algorithms which can all be seen as different strategies for traversing the graph of plans:

ESC (Exhaustive Search with Counting). Enumerates all nodes (plans) by indexing them with base- $(P \cdot S)$ numbers and mapping each digit to an action configuration per operator. Conceptually, ESC explores the entire graph of plans, evaluating each node independently and in parallel. It guarantees the globally optimal plan but is only feasible for small–medium search spaces.

Greedy (Greedy Search). Starts from the root plan and, at each iteration, examines all 1-hop neighbours (all single actions on all operators). It chooses the single best action (the neighbour with the best improvement according to the cost function), moves to that neighbour, and repeats. GSP builds a path in the graph of plans by always following the best one-step improvement. It is extremely fast but may get stuck in local optima.

RSS (Random Sampling Search). Uses the same enumeration scheme as ESC, but samples only a subset of nodes (plans) in the graph. It evaluates the sampled plans and returns the best among them. RSS trades optimality for controllable execution time and reduced risk of becoming trapped in local minima.

5.3 Introducing ESQ and Heuristic in the APEROL Suite

Exhaustive Search with a Queue (ESQ) The Exhaustive Search with a Queue (ESQ) algorithm explores the entire Graph of Plans that can be reached from a *root plan*. Conceptually, ESQ performs a breadth-first exploration of the graph of plans without exploiting APEROL's plan enumeration scheme. Note that an important distinction between ESQ and ESC is that the former always starts its exploration from the root plan, while the latter always starts from

Plan #0. Starting from the root plan, ESQ generates all one-hop neighbors by applying all applicable actions to all operators. Each newly generated plan is inserted into a queue if two conditions hold: 1. It does not correspond to a setup that has already been visited (the *same-setup* filter), 2. It is not provably worse than the currently best plan according to a cheap look-ahead test (the *look-ahead* filter). For each candidate plan, ESQ computes its total cost, which incorporates both the steady-state cost of executing the plan and the migration cost incurred when switching from the current root plan to that candidate. ESQ keeps track of the plan with minimum total cost and terminates when the queue becomes empty, i.e., when all reachable plans have been examined. ESQ is naturally parallel. The application of actions to different plans can be distributed across workers.

Algorithm 3 ESQ: Exhaustive Search with Queue over the graph of plans. **Input:** root workflow graph *root*, list of actions *actions*. **Output:** optimal plan *minPlan*.

Require: Workflow graph *root*, list *actions*

Ensure: Optimal plan *minPlan*

```

1:  $Q \leftarrow \{root\}$  ▷ queue of workflow graphs
2:  $V \leftarrow \{SIGNATURE(root)\}$  ▷ set/queue of workflow signatures
3:  $minPlan \leftarrow root$ 
4:  $minCost \leftarrow root.cost$ 
5: while  $Q \neq \emptyset$  do
6:    $g \leftarrow DEQUEUE(Q)$ 
7:   for all vertex  $v$  in  $g$  do
8:     for all action  $\alpha$  in actions do
9:       if  $v$  is not affected by  $\alpha$  then
10:        continue ▷ same-setup filter: skip if  $v$  already runs on the platform/site of  $\alpha$ 
11:      end if
12:       $APPLYACTION(g, v, \alpha)$  ▷ may update  $minPlan, minCost, Q, V$ 
13:    end for
14:  end for
15: end while
16: return  $minPlan$ 
17: procedure  $APPLYACTION(g, v, \alpha)$  ▷ uses globals  $minPlan, minCost, Q, V$ 
18:    $s \leftarrow SIGNATURE(g)$ 
19:    $s' \leftarrow MARK(s, v, \alpha)$  ▷ new signature by marking the choice (e.g., platform/site) for  $v$ 
20:   if  $s' \in V$  then
21:    return ▷ look-ahead filter: skip if this workflow has already been visited
22:   end if
23:    $g' \leftarrow g$  ▷ work on a copy of  $g$  (conceptually)
24:   apply  $\alpha$  on  $v$  in  $g'$  ▷ e.g., change placement/site/platform of  $v$ 
25:   update dependencies of  $v$  in  $g'$  ▷ modify all occurrences of  $v$  in adjacent vertices of  $g'$ 
26:    $c \leftarrow g'.COMPUTECOST()$ 
27:   if  $c < minCost$  then
28:      $minCost \leftarrow c$ 
29:      $minPlan \leftarrow g'$ 
30:   end if
31:    $ENQUEUE(Q, g')$ 
32:    $V \leftarrow V \cup \{SIGNATURE(g')\}$ 
33: end procedure

```

Heuristic Search (HEURISTIC) The Heuristic Search (HEURISTIC) algorithm trades completeness for efficiency by steering the exploration towards *promising* regions of the graph of plans. Instead of traversing all reachable plans, HEURISTIC only continues exploring from plans that lie on the current Pareto front with respect to multiple performance objectives (e.g.,

throughput, latency, communication cost, energy consumption).

Again, the search starts from the root plan. HEURISTIC generates the one-hop neighbours of the root by applying all actions to all operators. For each neighbor, it computes a multi-objective cost vector

$$\mathbf{c}(p) = (c_1(p), \dots, c_m(p))$$

and maintains the Pareto front of all examined plans. Only plans on this Pareto front are considered *promising* and inserted into the exploration queue. Plans that are dominated in all objectives by some other plan are considered unattractive and are not expanded further. Whenever a new promising plan is inserted into the queue, HEURISTIC recomputes the Pareto front, removing from the queue any plans that have just become dominated. This yields a multi-objective, BFS-like exploration that focuses the search on a small number of high-quality candidates, while still allowing the algorithm to discover plans that are multiple hops away from the root plan.

5.4 The DAG* Algorithm for Optimal Physical Plans

The DAG* algorithm we introduce in this section takes a totally different optimization approach. Instead of focusing on parallelization for swiftly exploring the search space, it concentrates on pruning a considerable amount of the search space by employing a heuristic cost, a real cost and an estimated cost function. DAG* [74] is an A*-like algorithm that swiftly outputs a physical execution plan that is guaranteed to be optimal only with respect to end-to-end latency, without exhaustively examining the entire search space. To achieve that, it topologically sorts the logical workflow and progressively examines physical instantiations to network devices for the involved operators (i.e., no complete root plan is accounted for). At each step n , the algorithm adds a new operator to the partial physical plan that has been built so far and computes an estimated cost computed as, figuratively speaking, $g(n) = f(n) + h(n)$, where $f(n)$ is the current cost of the partial physical plan and $h(n)$ is a heuristic cost expressing an under-estimation of how much cost will be added to the partial plan to make a full plan, i.e, with all operators assigned to devices/platforms. Then, the partial plan is inserted into a priority queue, sorted on $g(n)$. The first partial plan in the queue is then dequeued and the algorithm attempts to expand it by adding a new operator as described above. When the dequeued plan is a full physical plan, the algorithm concludes and outputs the optimal plan found. To output a physical workflow execution graph, instead of a path (that is the output of the original A* algorithm), DAG* imposes two important rules: (i) at each step, only one operator can be examined for physical instantiations to expand a dequeued partial plan, and (ii) no logical operator can be instantiated to any physical implementation unless all of its upstream operators have been included in the currently examined partial plan.

5.4.1 DAG* New Problem Formulation

Consider a workflow as a *weighted Directed Acyclic Graph (DAG)* $\mathcal{G} = (V_{\mathcal{G}}, E_{\mathcal{G}})$, where $V_{\mathcal{G}}, E_{\mathcal{G}}$ are the sets of nodes and edges accordingly. Each node $i \in V_{\mathcal{G}}$ has a set of possible configurations denoted as C_i , where each configuration $c_k^i \in C_i$ is related with a cost $\text{conf}_c(c_k^i) \geq 0$. Note that $\forall i, j \in V_{\mathcal{G}}$, where $i \neq j$, the sets C_i and C_j may differ, while if a specific configuration c_k exists in both sets, then the corresponding cost of c_k may vary for each node, i.e., $\text{conf}_c(c_k^i) \neq \text{conf}_c(c_k^j)$. Moreover, each edge $e_{i \rightarrow j} \in E_{\mathcal{G}}$ has a non-negative

Algorithm 4 Heuristic Search with Queue over the graph of plans

Require: Graph $root$, list $actions$

Ensure: Optimal plan $minPlan$

```

1:  $Q \leftarrow \{root\}$ 
2:  $V \leftarrow \{SIGNATURE(root)\}$ 
3:  $minPlan \leftarrow root$ 
4:  $minCost \leftarrow root.cost$ 
5: while  $Q \neq \emptyset$  do
6:    $g \leftarrow HEAD(Q)$ 
7:   for all vertex  $v$  in  $g$  do
8:     for all action  $\alpha$  in  $actions$  do
9:       if  $v$  is not affected by  $\alpha$  then
10:        continue
11:      end if
12:       $APPLYACTION(g, v, \alpha)$ 
13:    end for
14:  end for
15: end while
16: return  $minPlan$ 
17: procedure  $APPLYACTION(g, v, \alpha)$ 
18:    $s \leftarrow SIGNATURE(g)$ 
19:    $s' \leftarrow MARK(s, v, \alpha)$ 
20:   if  $s' \in V$  then
21:    return
22:  end if
23:    $g' \leftarrow g$ 
24:   apply  $\alpha$  on  $v$  in  $g'$ 
25:   update dependencies of  $v$  in  $g'$ 
26:    $g'.COMPUTECOSTPEROBJECTIVE()$ 
27:   if  $Q.ISPARETOOPTIMAL(g')$  then
28:     $c \leftarrow g'.COMPUTECOST()$ 
29:    if  $c < minCost$  then
30:       $minCost \leftarrow c$ 
31:       $minPlan \leftarrow g'$ 
32:    end if
33:  end if
34:   $ENQUEUE(Q, g')$ 
35:   $Q.COMPUTENEWPARETOFRONT()$ 
36:   $V \leftarrow V \cup \{SIGNATURE(g')\}$ 
37: end procedure

```

▷ queue of workflow graphs
▷ signatures of visited workflows

▷ same-setup filter: skip if v already runs on the platform/site of α

▷ uses globals $Q, V, minPlan, minCost$

▷ new signature by marking the choice (e.g., platform/site) for v

▷ look-ahead filter: skip if this workflow has already been visited

▷ work on a copy of g

▷ e.g., change placement (site/platform) of v

▷ modify occurrences of v in all adjacent vertices of g'

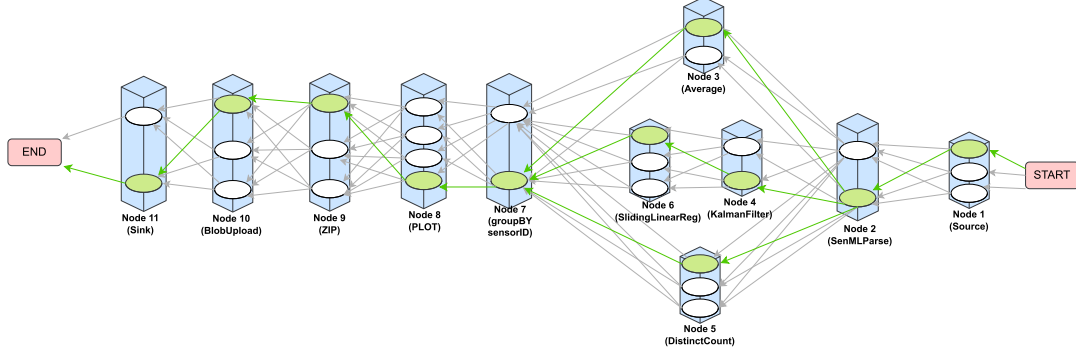


Figure 48: Example of DAG* output on the STATS workflow [69]. Nodes are the operators of the analytics workflow. Disks in each node are possible configurations, i.e., executing the operator represented by Node i on network device d_k . Each disk has a different cost (processing latency). An edge denotes operator dependency with a different cost weight (network latency) depending on the configurations they connect. Green-colored edges and disks compose the optimal graph returned by DAG*.

weight that depends on the configurations that have been assigned to nodes i and j , denoted as $edge_c(c_k^i, c_l^j)$. On top of that, each node i has a set of predecessors $Pr(i)$, i.e., set of nodes that are connected with i via an edge $e_{j \rightarrow i}$, where $j \in Pr(i)$; and a set of successors $Su(i)$, i.e., set of nodes that are connected with i via an edge $e_{i \rightarrow j}$, where $j \in Su(i)$.

Also, given $\mathcal{G}$, we insert two *auxiliary nodes*, namely the START and the END nodes, to guide the initial and final steps of our algorithm (see Sections 5.4.2 and 5.4.2). In particular, we connect the START node with each node $i \in V_{\mathcal{G}} : Pr(i) = \emptyset$ via an edge $e_{START \rightarrow i}$; while we connect the END node with each node $i \in V_{\mathcal{G}} : Su(i) = \emptyset$ via an edge $e_{i \rightarrow END}$.⁹

Illustrative Example: Consider the workflow graph of Figure 48. It includes interconnected nodes (workflow operators) that are executed to accomplish a higher order statistic computation analytic task. The interconnection between operators are the edges denoting the dependency of an operator on its previous ones. The workflow continuously ingests input data from Node 1 which is a Source operator, and each subsequent operator in the topologically sorted workflow continuously receives input from one or more other operators. Notably, a workflow can include both splits (the SenMLParse operator gives input to more than one other) and merges (the groupBy sensorID operator receives input from more than one operator). Each operator should be executed on some, among several network devices, though it has multiple device choices, termed configurations, with different costs/processing latencies ($conf_c$ in our problem formulation) represented as disks in Figure 48. Data can be transferred among the devices via $edge_c$ with a communication cost/latency equal to the weight of $edge_c$. The optimal graph of DAG* will have assigned each operator at a device aiming to minimize the entire cost (processing and communication latency) of the workflow as a whole.

DAG* finds the optimal graph based on some cost function. The solution that our algorithm will produce must contain $\forall i \in V_{\mathcal{G}}$, which should be assigned at a configuration $c_k^i \in C_i$.

⁹All edges that begin from START and all edges that arrive at the END have a zero cost (or weight).

Definition 25 (ANOC (AllNodesOneConfiguration) Graph). *A graph $\mathcal{G}'$ is called ANOC if and only if the following three conditions hold: (a) $V_{\mathcal{G}'} = V_{\mathcal{G}}$; (b) one configuration is assigned to each node $i \in V_{\mathcal{G}'}$; and (c) all nodes $i \in V_{\mathcal{G}'}$ are chosen to be connected via a unique edge with each node $j \in Pr(i)$.*

Simply put, an ANOC graph is one that contains all nodes, one chosen configuration per node and one edge connecting chosen configurations of connected nodes. Notably, this definition is easily extensible to broader scenarios, where, for instance, a node can be replicated with multiple configurations. In that case, the said node just needs to be replicated in the graph $\mathcal{G}$.

Definition 26 (Optimization Problem). *Given $\mathcal{G} = (V_{\mathcal{G}}, E_{\mathcal{G}})$, DAG^* outputs the optimal graph, $\mathcal{G}^* = (V_{\mathcal{G}}^*, E_{\mathcal{G}}^*)$ such that:*

$$\textbf{Minimize: } Real(\mathcal{G}^*) + \underbrace{h(\mathcal{G}^*)}_{=0} \quad (11)$$

Subject to: $\mathcal{G}^*$ is an ANOC graph

with the $Real(\mathcal{G}^)$ and the heuristic $h(\mathcal{G}^*)$ costs explained in Equation 13 and Equation 12, later on.*

We further point out that $h(\mathcal{G}^*)$ is the heuristic cost defined in Section 5.4.2. As explained later in Equation 12, this heuristic cost is zero for the END operator of any given $\mathcal{G}$, i.e. just at the point when DAG^* reaches the optimal solution for the entire workflow. We superfluously include $h(\mathcal{G}^*)$ (explicitly zeroing it out) in the above definition for completeness and for emphasizing the connection with Section 5.4.2.

5.4.2 The Proposed DAG^* Algorithm

Nodes' Heuristic Cost DAG^* , similar to the classic A*, utilizes a heuristic cost function for each node in $V_{\mathcal{G}}$ to guide its search. Remarkably, DAG^* contributes with a novel formula for the heuristic cost of each node $i \in V_{\mathcal{G}}$, denoted as $h(i)$. This new formula is a prerequisite in order to operate aiming for optimal graphs instead of optimal paths. For a given workflow, the heuristic cost should be an optimistic estimation on what is the cost (aggregate network and processing latency) from a given node/operator (see Figure 48) to reach the END. For a heuristic to be admissible, i.e., guarantee optimality of DAG^* , $h(i)$ should always be an underestimation of the real cost from a given node/operator to reach the END. The novel, admissible heuristic introduced by DAG^* for optimal graph discovery is given by the following formula:

$$h(i) = \begin{cases} 0 & \text{if } i \in \{Pr(END), END\} \\ \max_{j \in Su(i)} \{h(j) + \min_{k \in C_j} \{conf_c(c_k^j)\}\} & \text{else} \end{cases} \quad (12)$$

We stress that the computation of the heuristics is performed in a reverse topological sort of the nodes; it starts with the END node,¹⁰ proceeds with the $Pr(END)$ set, then the nodes in $Pr(Pr(END))$ and so on. This process terminates by computing the heuristic of the START. Note also that Equation 12 leads to an *admissible* (heuristic) function since it never overestimates the real cost from any node $i \in V_{\mathcal{G}}$. This holds because, for each node i , Equation 12 computes the sum of the minimum configuration costs and the heuristics for

¹⁰Since the END node is a 'goal-node', its heuristic cost is equal to zero. Since it is a virtual node, it has a zero configuration cost as well.

each j , where $j \in Su(i)$. Then, in case of a node with multiple successors (e.g. Node 2 in Figure 48) we set the heuristic of i as the maximum computed sum of the minimums of its successors. This guarantees that our heuristic never overestimates the actual cost, since our approach attempts to discover the optimal graph, so all nodes are always contained in the solution. We highlight that we select the maximum of the minimums instead of the minimum of the minimums, which could also be used, since it offers a less (but still) optimistic and more realistic estimation of the actual cost. As such, the maximum of minimums gives an admissible heuristic simultaneously guiding our algorithm faster to the optimal solution. Now, for num_conf possible configurations per node, the $h(i)$ values can be precomputed using $\mathcal{O}(|V_G| \cdot (num_conf + B))$ time, where B is the maximum number of successors $Su(i)$ of any node, by first computing the quantities $\min_{k \in C_j} \{conf_c(c_k^j)\}$ in $\mathcal{O}(num_conf)$ time per node.

Illustrative Example: Consider that we want to compute the heuristic of Node 2, i.e., $h(2)$. We have that $Su(2) = \{\text{Node 3, Node 4, Node 5}\}$ with assumed heuristics $h(3) = 10$, $h(4) = 8$ and $h(5) = 12$. Moreover, for ease of exposition, assume that Node 3 and Node 5 have only one valid configuration with costs 5 and 1 accordingly, while Node 4 has two valid configurations with costs 5 and 7. Thus, from Equation 12 we get that $h(2) = \max\{\{h(3) + 5\}, \{h(4) + \min\{5, 7\}\}, \{h(5) + 1\}\} = \max\{\{15\}, \{13\}, \{13\}\} = 15$.

DAG* also accounts for a real cost for each (partial or complete) graph of $\mathcal{G}$ that it has examined at any given point in time. This real cost, formally defined in the next section, for each operator of a given workflow, measures the cost from the START to the said node/operator. The real cost depends on (a) employed configurations for the visited nodes; (b) the weight of the edges that connect these configurations and (c) a chosen aggregation function.

Partial Graphs' Cost Since our goal is to return the optimal graph, our algorithm extends partial graphs instead of just nodes. As we explain in Section 5.4.2, DAG* maintains a priority queue where partial graphs are stored along with their corresponding costs. Each partial graph g contains the so far visited nodes and used edges of $\mathcal{G}$ and has some cost based on the configurations that have been assigned to its nodes. The *estimated cost* of a partial graph g is the sum of the real and the heuristic costs, formally:

$$f(g) = \underbrace{conf_c(c_k^i) + \underbrace{\text{aggr}_{\substack{i \in T_g | c_k^i \\ j \in Pr(i) \in g^- \\ \ell \in C_j}} \{Real(g^-) + edge_c(c_k^i, c_\ell^j)\}}_{Real(g)} + \underbrace{\min_{i \in T_g} \{h(i)\}}_{h(g)} \quad (13)$$

where T_g is the set of the *terminal nodes* of partial graph g , i.e., those nodes with empty $Su()$ sets. The real cost $Real(g)$ is computed from the cost of g^- , which is a dequeued, visited partial graph. We add the cost of a new configuration for $i \in V_g$ and we also add the weight of the edges that connect this configuration (disk in Figure 48) to the configurations of its predecessors in g^- . We aggregate across all these edges using any function, e.g., sum, max, etc., depending on the application, i.e., serial execution may imply sum, while parallel execution may imply max.

Illustrative Example: Consider the scenario where $aggr = max$ and we just dequeued g^- which contains all Nodes 1 to 6 with a cost of $Real(g^-) = 56$. Now, we can extend g^- by inserting Node 7, which has a heuristic $h(7) = 8$, into one of its valid configurations. Assume that we generate g by selecting c_1^7 which has a cost of 4. Moreover, we assume that based on their applied configurations the communication costs from Nodes 3, 5, and 6 to 7 are equal to 3, 1, and 2 accordingly. Thus from Equation 13 we get that $f(g) = 4 + max\{\{56 + 3\}, \{56 + 1\}, \{56 + 2\}\} + 8 = 4 + 59 + 8 = 71$.

Extending a Partial Graph DAG* uses an estimated cost-based priority queue to store partial graphs, during its quest of reaching the optimal graph it seeks for. Here, we describe the partial graph extension operation of our approach, as we visit more nodes towards optimal, complete graph discovery. To extend any partial graph g , g should be contained in a priority queue, denoted as Q , and should have the smallest estimated cost (Equation 13). Then g is selected and removed from Q . DAG* introduces two novel rules in order to ensure that the algorithm will output a graph instead of a path. The extension of partial graph g must follow these two rules:

Rule 1 During an extension, only one node i can be inserted to g along with the edges that lead to i and exist in $\mathcal{G}$, i.e., $\forall j \in Pr(i) : e_{j \rightarrow i} \in E_{\mathcal{G}}$.

Rule 2 A node i can be selected if and only if all nodes in $Pr(i)$ have been already inserted in g , i.e., $\forall j \in Pr(i) : j \in V_g$.

Rule 1 says that we must expand g selecting any, but one at a time, node i and Rule 2 says that, for a node to be qualified for selection, all its predecessors must have been included in g . Once we select i , we should generate $|C_i|$ (cardinality of C_i) new partial graphs that differ in the employed configuration of the newly inserted node i . Subsequently, we compute their estimated cost and we insert each of them into Q . For each expansion, the complexity for computing the costs of each of the $|C_i|$ new partial graphs is $\mathcal{O}(|Pr(i)| \cdot num_conf)$.

Lemma 1. [74] *For any partial graph g that is stored in Q , it holds that $\forall i \in V_g$ all nodes in sets $Pr(i), Pr(Pr(i)), \dots$ also exist in V_g , i.e., no node i can exist in any partial graph $g \in Q$ without all the nodes j that are connected with i , via an edge $e_{j \rightarrow i}$, in $\mathcal{G}$.*

Illustrative Example: In the initial iteration, DAG* inserts to Q a partial graph g_0 that contains only the START node. g_0 is then dequeued and extended. In Figure 48, this extension process will enqueue 3 partial graphs, because Node 1 has 3 possible configurations (disks). These partial graphs will be prioritized based on their estimated cost (Equation 13). This process will continue in a similar way. According to Rule 2, no extension will include Node 7 until a partial graph that contains all Nodes 3, 5 and 6 is dequeued. DAG* terminates only when an ANOC graph is dequeued.

Algorithm 5 provides the DAG* pseudocode. Initially, nodes (START, END) are inserted into the given graph $\mathcal{G}$ (Line 1), and the heuristics of each node are computed (Line 2) based on Equation 12. In Line 4 we insert a partial graph that contains the START node. Now, the loop in Line 7 contains the core idea of DAG*. In detail, the partial graph with the lowest estimated cost is dequeued. If the dequeued partial graph is an ANOC graph (Lines 9-12), then we will terminate our search returning the optimal solution (Line 22). This is because we will have a complete graph with lower real cost than any of the estimated (best case) costs

Algorithm 5 DAG* algorithm for optimal graph discovery

Input: Graph $\mathcal{G}$, Configuration costs $conf_c$, Edge costs $edge_c$, Priority queue Q (default $Q = \emptyset$)

Output: Optimal graph $\mathcal{G}^*$

```

1: Insert the START and END nodes at  $\mathcal{G}$ 
2: computeHeuristics( $\mathcal{G}$ ,  $conf_c$ )
3: if  $Q = \emptyset$  then
4:    $Q.enqueue(START)$ 
5: end if
6:  $\mathcal{G}^* = \emptyset$ 
7: while  $Q \neq \emptyset$  do
8:    $g = Q.dequeue()$ 
9:   if  $g$  is ANOC then
10:     $\mathcal{G}^* = g$ 
11:    break
12:   end if
13:    $candidates = nodesMeetingRule2(\mathcal{G}, g)$ 
14:    $newNode = rand(candidates)$ 
15:                                     ▷ Apply Rule 1
16:   for all possible confs of  $newNode$  do
17:      $tmpG = insertNode(newNode_i, \mathcal{G}, g, conf_c, edge_c)$ 
18:      $computeCost(tmpG)$ 
19:      $Q.enqueue(tmpG, sortKey = estimated\ cost)$ 
20:   end for
21: end while
22: return  $\mathcal{G}^*$ 

```

of the (partial graphs) remaining in the queue (see Section 5.4.3). If the dequeued partial graph was not an ANOC, then in Line 13 we find the nodes that are qualified for selection in the expansion of the dequeued partial graph based on Rule 2. In Line 14 we choose, among these candidates, one candidate node that will be inserted into the partial graph (Rule 1). Then, once we select the node that will be inserted, the inner loop (Line 16), for every available configuration (of the selected node), will (a) insert the node (assigned to a specific configuration) to the dequeued partial graph; (b) compute its estimated cost based on Equation 13; and (c) insert the extended partial graph to the priority queue in the appropriate index based on its estimated cost. Finally, the algorithm will proceed to the next iteration.

In the worst case, the number of iterations-expansions of DAG* is $\sum_{i \in V_{\mathcal{G}}} num_conf^{|\mathcal{X}(i)|} = \frac{num_conf^{|V_{\mathcal{G}}|+1}-1}{num_conf-1} = \mathcal{O}(num_conf^{|V_{\mathcal{G}}|})$. The running time cost of each expansion was presented in Section 5.4.2.C. Also, in the worst case $\mathcal{O}(num_conf^{|V_{\mathcal{G}}|})$ partial graphs will be enqueued in the priority queue, but the memory footprint of each such partial graph is small, because we do not need to explicitly materialize them. Each partial graph can instead be represented by the IDs of the nodes that belong to it, along with a number for each node i.e., for the selected configuration for that node [34].

5.4.3 On DAG* Optimality

Note that due to the way we insert the auxiliary START, END nodes and since the solution to this optimization problem is an ANOC graph, it holds that $T_{\mathcal{G}^*} = \text{END}$. However, as described in Section 5.4.2, we know that $h(\text{END}) = 0$. As such, we can remove the second term of Equation 11, since it is always equal to zero due to the existing constraints for all valid

solutions. Now, we prove theoretically that DAG* is able to discover the optimal graph for our optimization problem.

Theorem 6. [74] *The ANOC graph, g , that DAG* returns is the one with the lowest cost, i.e., it is the optimal graph.*

5.4.4 Dynamic Setting

Many real-world scenarios are characterized by dynamic changes that occur unpredictably over time. In IoT settings network devices may lose connectivity or new ones may subscribe to the network [81]. In the analytics scenarios considered in this work, the volume and velocity of continuously ingested data may vary over time. Therefore, it is crucial for the employed algorithms to be able to adjust to these challenging settings by efficiently handling the changes that may alter configuration cost values. Given this motivation, we introduce an incremental version of DAG*, called d-DAG*.

To accommodate such dynamic cost changes efficiently, d-DAG* must undertake the task of updating its already-computed priority queue, Q ,¹¹ ensuring the removal of nodes containing outdated information. Thus, given a change, we have to find the affected nodes and discard them from any partial graph $g \in Q$. In detail, the occurred change affects directly the node i , i.e., the node that its configuration(s) cost(s) just changed. As a result, the node i has to be discarded from any partial graph $g \in Q$. Moreover, since no partial graph will contain node i , due to Rule 2 (see Section 5.4.2), d-DAG* has to also remove the nodes that exist in sets $Su(i), Su(Su(i)), \dots$ to the END node, from all partial graphs in Q . On top of that, such change could also affect j 's heuristic function, $h(j)$, where $j \in Pr(i)$ (see Equation 12). In such case, the change could similarly 'propagate' further, affecting the heuristics of the nodes in sets $Pr(Pr(i)), Pr(Pr(Pr(i))), \dots$ to the START node. Once we check which of these nodes were affected, denoted as Aff , we also ensure that the post-change queue satisfies Lemma 1. As such, we discard all the nodes that exist in sets $Su(j), Su(Su(j)), \dots$ to the END node, where $j \in Aff$.

Having discarded all the affected nodes from all partial graphs in Q , we compute the partial graphs estimated costs based on Equation 13. We then sort them with respect to the updated estimated costs and we remove the potential duplicates that may have been created. As a result of the aforementioned process, we have generated an updated priority queue by exploiting the information that had been already computed during the execution of DAG* before the change occurrence. Thus, we achieve the desired incremental version of our algorithm since our approach can exploit Q without necessarily starting from scratch to find the post-change optimal solution.

Algorithm 6 provides the d-DAG* pseudocode. Initially, we insert to the set of the affected nodes, denoted as Aff , the node i affected by the change (Line 4). In Line 5 we use a recursive function to find the nodes $Su(i), Su(Su(i)), \dots$ and update Aff set. Then, if the occurred change affected the heuristic of node i we utilize a recursive function to find which of the nodes in $Pr(i), Pr(Pr(i)), \dots$ were also affected and update Aff set (Lines 7 - 10). Now, given the Aff set, we have to discover the successors of j , where $j \in Aff$, in order to satisfy Lemma 1 and update the Aff set (Lines 11 - 13). Next, after the prescribed updates,

¹¹Note that initially the DAG* has been employed in an initial setting, returning the optimal graph through a priority queue Q that had been created (see Section 5.4.2).

the Aff set may contain duplicates which we can remove without affecting the result of the algorithm. Given the computed Aff set, we iterate through Q in order to remove from any stored partial graph the nodes that were affected and re-compute their estimated cost, which will be used as 'key' in order to sort the resulted priority queue Q_f (Lines 15 - 19). Once, we create the Q_f we remove any potential duplicate partial graphs that may have been created due to the node removal process described earlier. Finally, in Line 21, using the Q_f priority queue we call the Algorithm 5, i.e., the DAG* algorithm, which will not begin from scratch (an empty priority queue). In the worst case, d-DAG* will have to update all the affected partial graphs from the priority queue which are up to $num_conf^{|V_G|}$; and call DAG* in Line 21 of Algorithm 6 with $Q_f = \emptyset$. Thus, the search will start from scratch and the complexity of d-DAG* will in the worst case be $O(num_conf^{|V_G|})$.

Algorithm 6 d-DAG* algorithm

Input: Graph G , Priority queue Q , Configuration costs $conf_c$, change ch_i , old heuristic h , Priority queue $Q_f = \emptyset$

Output: Optimal graph G^*

```

1:  $h' = \text{computeHeuristics}(G, conf_c)$  ▷ Update affected heuristics
2:  $Aff = \emptyset$ 
3:  $i = ch_i.\text{getAffNode}()$ 
4:  $Aff.add(i)$ 
5:  $su = \text{findSuccessors}(i)$ 
6:  $Aff.add(su)$ 
7: if  $h(i) \neq h'(i)$  then
8:    $pr = \text{findPredecessors}(i)$ 
9:    $Aff.add(pr)$ 
10: end if
11: for  $j \in Aff$  do
12:    $Aff.add(\text{findSuccessors}(j))$ 
13: end for
14:  $Aff.removeDuplicates()$ 
15: for  $g \in Q$  do
16:    $g.delete(Aff)$ 
17:    $\text{computeCost}(g)$ 
18:    $Q_f.enqueue(g, \text{sortKey} = \text{estimated cost})$ 
19: end for
20:  $Q_f.removeDuplicates()$ 
21:  $G^* = \text{DAG}^*(Q_f)$  (run Algorithm 5 from Line 3)
22: return  $G^*$ 

```

On d-DAG* Optimality Here we prove theoretically that d-DAG* is able to discover the optimal graph. For this section, we denote as Q the priority queue that DAG* has created in the initial setting, the first time it runs, and as Q_f the priority queue that d-DAG* generated after a change that occurred in the setting (Algorithm 6).

Lemma 2. [74] *For any given partial graph g , if Lemma 1 is satisfied then we can always construct an ANOC graph, which contains the configurations of nodes that are already contained in g with any combination of configurations of the nodes that have not inserted to g yet.*

Lemma 3. [74] *All partial graphs $g \in Q_f$ satisfy Lemma 1.*

Based on the operation of DAG* (i.e., when it extends a node j it adds $|C_j|$ equal,¹² partial

¹²We say that partial graphs g_1 and g_2 are equal if and only if they have the same heuristics, $V_{g_1} = V_{g_2}$,

graphs to the queue, where each partial graph contains a different configuration of j), we can guarantee that at timestep t , where the optimal graph is dequeued, for each node i it holds that $\exists g \in Q : \forall c_k^i \in C_i$.

Now, assume that a change occurred in our setting that affected a set of nodes. Note that this set of nodes is computed by d-DAG* which creates the Q_f . In detail, d-DAG* does not delete any of the partial graphs stored in Q , but it replaces some of the partial graphs with one partial graph that does not contain any affected node. Thus, we cannot have a partial graph $g \in Q_f$ that contains only a subset of some node's configurations, because if a partial graph with a configuration $c_k^i \in C_i$ was affected, then all the 'equal' partial graphs of g that differ only to i 's configurations will be affected as well. As a result $\forall i$ it holds either (a) $\exists g \in Q_f$ for $\forall c_k^i \in C_i$; or (b) $\nexists g \in Q_f$ for $\forall c_k^i \in C_i$.

Theorem 7. [74] *Given Q_f there is no ANOC graph that cannot be reached.*

Given Theorems 6 and 7, by employing DAG*, not from scratch, but from Q_f as initial point, we will definitely reach the optimal solution.

Lemma 4. *The ANOC graph, g , that d-DAG* returns is the one with the lowest cost, i.e., it is the optimal graph.*

5.5 Experimental Evaluation

5.5.1 Objectives, Metrics and Workload

We compare the APEROL suite and DAG* algorithms against two performance criteria. The performance of the output optimization plan and the Average Algorithm Execution Time per decision. For APEROL, the goodness of the output plan is measured in Plan Cost Improvement, which corresponds to how much one or more adaptation decisions are improving the root plan as each workflow is executed, i.e., $GoP_\pi - GoP_{root}$. For DAG*, which operates only on the latency performance dimension, the performance of the output plan is measured in Latency i.e. time units in milliseconds.

For the Plan Cost Improvement of APEROL we first derive the value of each dimension of the root plan and each candidate plan. Then for each cost dimension we compute the improvement ratio of the candidate plan over the root plan. Finally, we multiply each dimension with a constant factor to bring them in a common range. The Average Algorithm Execution Time shows how fast can each algorithm devise a new plan, which is critical for timely action upon performance degradation. In that, we can examine the plan quality vs rapidness of the optimization decision trade-offs.

We employ two well known benchmarks, namely the Yahoo Benchmark [9, 35] and the RIoT benchmark [69]. The Yahoo Benchmark is a standard stream analytics benchmark composing a workflow of selection, projection, join, windowing and aggregation operators (8 operators in total). RIoT [69] provides four real-world workflows from the smart cities domain [69] – termed TRAIN, PRED, STATS, ETL – with 8, 9, 11, 11 operators, respectively. The RIoT workflows involve a machine learning training, predictive analytics, higher order statistic extraction and

$E_{g_1} = E_{g_2}$ and all nodes in g_1 and g_2 have the same configuration except one node which has a different configuration in g_1 and g_2 .

extract-transform-load task, respectively. The interconnections between the operators of the aforementioned workflows can be visualized in Figures 49, 50, 51, 52, 53.

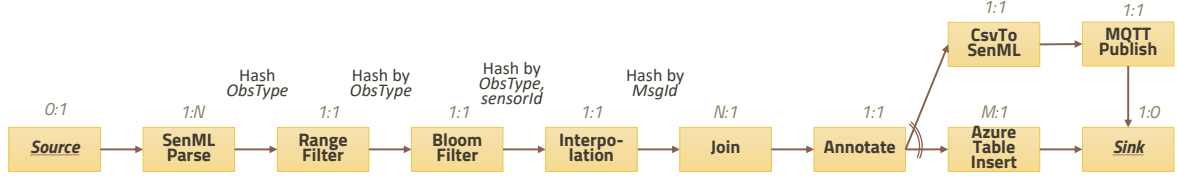


Figure 49: ETL Workflow

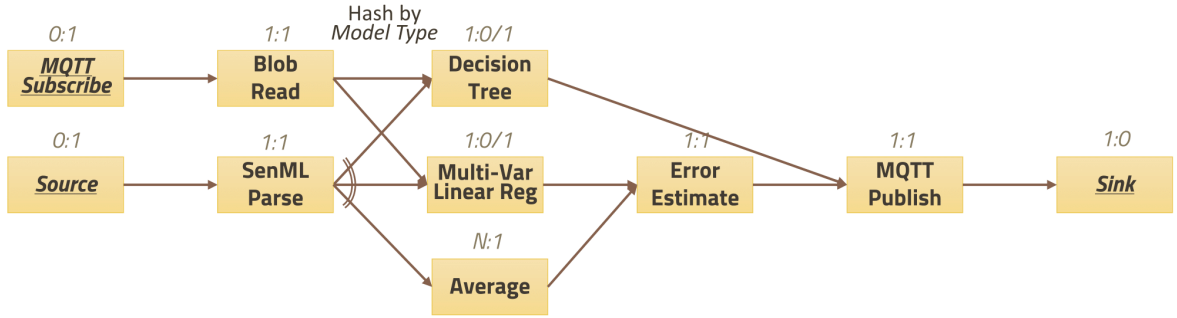


Figure 50: PRED Workflow

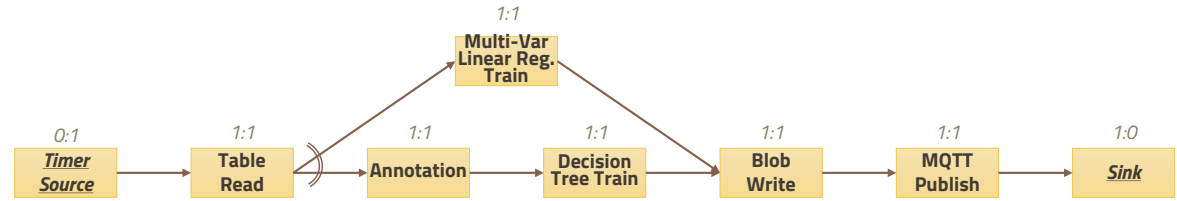


Figure 51: TRAIN Workflow

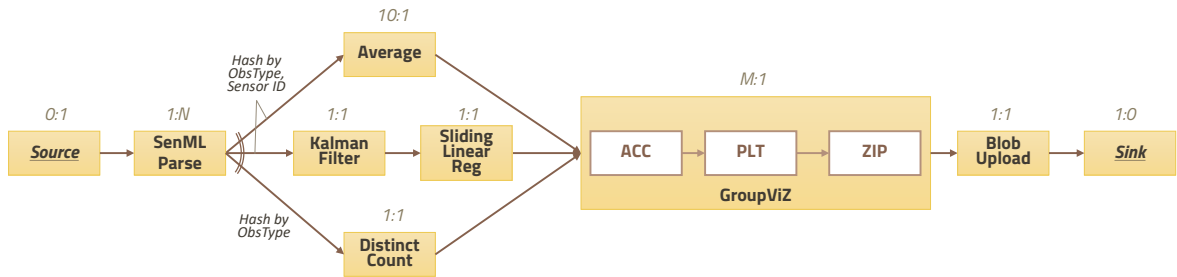


Figure 52: STATS Workflow

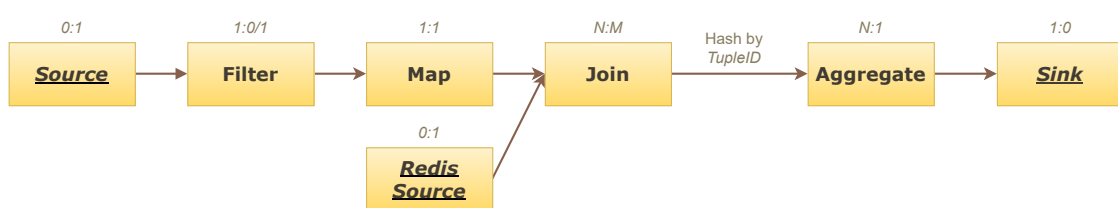


Figure 53: YAHOO Workflow

We employ different network setups including both a real testbed from the publicly available

FIT IoT-LAB [1] and simulated network setups. We describe the utilized network setups in more detail.

5.5.2 Real Testbed Configuration

We reserve all the 194 available devices at the Grenoble site of the publicly available FIT IoT-LAB [1] testbed, consisting of Raspberry Pi 3 boards and A8 devices. The A8 board is based on an ARM Cortex-A8 micro-processor. Both of these types of devices ship with an embedded Linux operating system. To derive the network topology, exhaustive all-to-all round-trip latency measurements were performed across the reserved nodes, and the results were used to construct the corresponding connectivity graph. This approach provides a precise and empirical view of the physical interconnections and communication delays within the testbed, which is essential for accurately modeling operator placement costs. In order to create variable-sized networks (of up to 194 devices), subgraphs of the full connectivity graph were extracted. These subgraphs were selected to maintain representative structural characteristics (e.g., branching factor, link diversity, and node heterogeneity), ensuring that smaller configurations preserve the statistical properties of the full testbed. On this Real Testbed, Raspberry Pi 3 nodes can execute functional Python code and JavaStreams and can even host a lightweight Apache Flink deployment with a single Task Manager instance. The resource-constrained A8 nodes are limited to running only functional Python or JavaStreams, due to limited hardware capabilities. We monitor setups composed of {7, 15, 31, 127, 194} sites in the Real Testbed. This Real Testbed can only execute the Yahoo Benchmark workflow due to resource constraints, but it is used in our experiments for validating the trends seen in simulated networks (see below) and, ultimately, for comparing the performance of APEROL against DAG* [74] and other state-of-the-art approaches in the literature including NEMO [7] and Governor [8] on an actual network, besides simulated setups. Figure 54 illustrates the network of 194 devices, as all others have similar topologies to the simulated networks that are going to be presented in the following sections. In this case, the binary tree topology -which is common in all other networks of our experimental evaluation- is not complete, and therefore some of its paths from the root node (i.e. the cloud) to the leaves of the tree will have different lengths than others. As also shown in the following figures, we also showcase the existence of gateway devices by coloring some extra nodes purple, and sensors by coloring these nodes red.

5.5.3 Simulated Networks Configuration

For our simulated network setups of sizes {7, 15, 31, 127, 1023, 2047} sites, we use the highly cited iFogSim tool [28, 46] to simulate both devices of various computational capacities and RIoT workflows' execution over these networks. Figure 55 illustrates the six simulated networks that are employed in our experiments. We should note that the topology of the networks is the same both for the heterogeneous and the homogeneous setup. The differentiating factor between these setups is the distribution of the devices' computational capacity. For instance, in the homogeneous setup, the networks consist of Raspberry Pi-class devices, while in the second setup, the heterogeneity is increased by the introduction of various other devices with different computational capacities.

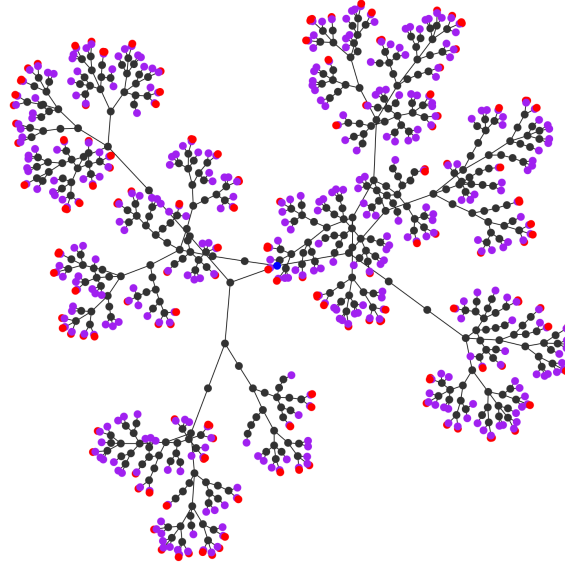


Figure 54: Visual illustration of the Real Testbed

Simulated Heterogeneous Network Setup The devices of this setup consist of a mix of resource-constrained edge nodes such as ESP32-based sensors, Raspberry Pi 4, Jetson Nano boards, and high-end edge servers. Execution engines match to device capabilities: MiNiFi C++ runs on light-weight devices and MiNiFi Java runs on Raspberry Pi-class devices, while Kafka Streams and Apache Flink run on more powerful nodes like Jetson Xavier or Intel NUCs. This setup is useful to judge the effectiveness of each APEROL algorithm in cases when some actions can make an important (positive or negative) difference compared to others and, therefore, algorithms susceptible to local optima, may underperform.

Simulated Homogeneous Network Setup In this setup, all nodes are Raspberry Pi-class devices. A single runtime engine across all nodes for processing and edge-to-cloud data forwarding is deployed. Due to the homogeneous setup, algorithms (HEURISTIC, GREEDY) that are more susceptible to local optima are less likely to underperform in the quality of the returned plan. Given the number of operators per workflow cited previously, this means that the cardinality of the search spaces we experiment with starts from 8^7 and exceeds 11^{2047} candidate execution plans.

5.5.4 Algorithmic Setup & Hardware

The APEROL algorithms run with a default parallelism of 4, which is a sweet spot (see Section 5.5.7, where we also vary parallelism). DAG* at its current stage is not parallelizable, therefore its parallelism is set to 1. We run all our experiments on a MacBook Air M2 (8 Cores/16GB RAM), but in Section 5.5.7, we also report on APEROL's performance on a server Intel(R) Xeon(R) Silver 4310 CPU @ 2.10GHz, 20 cores/40 threads, and 256Gb RAM. We set a timeout for ESQ and ESC APEROL algorithms equal to the average execution time of RSS, HEURISTIC, GREEDY. After the timeout, ESQ and ESC output the best plan found so far. For RSS we set a default sample size of 3K execution plans and we perform Sobol sequence-based sampling. We term this version as RSS-3. Sobol sequence-based

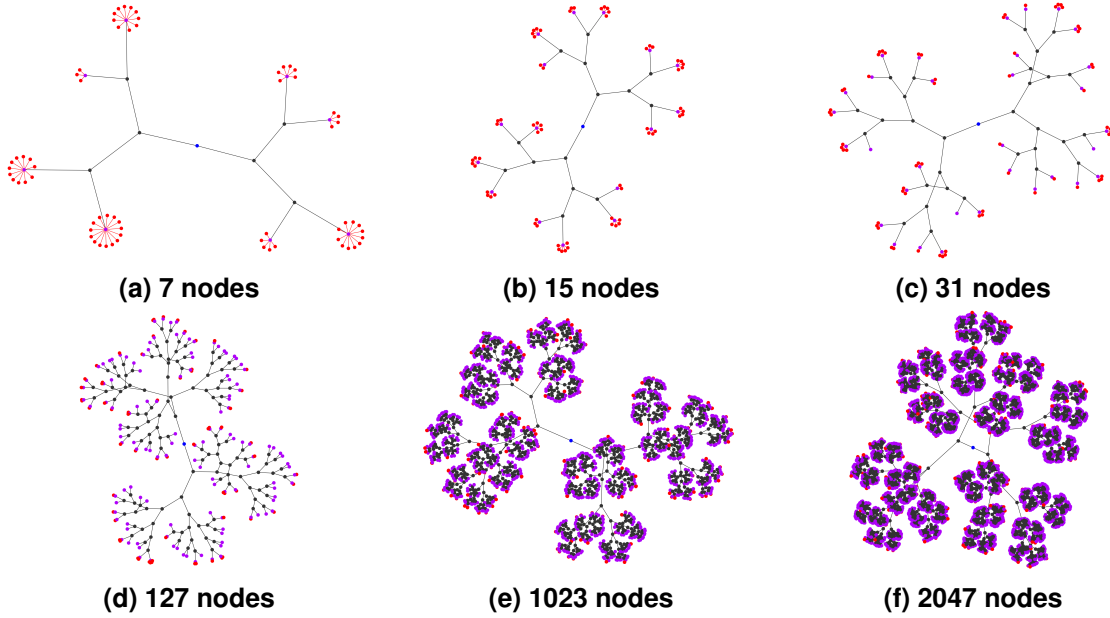


Figure 55: Visual illustration of the simulated networks that are used in the experimental evaluation

sampling allows RSS to evenly sample the search space irrespective of its size, which is exactly the differentiating factor, given the timeout for the exhaustive algorithms in APEROL. We experimented with sample sizes of 5K, 7K and 10K, but we found little difference in plan costs and algorithm execution times. For GREEDY, the introduction of thread safe data structures and synchronization mechanisms results in overheads, which deteriorate the otherwise minimal Algorithm Execution Time. Therefore, we set the parallelism of GREEDY to 1.

Main findings ① The APEROL and DAG* algorithms are complementary in their utility in different settings. ② In heterogeneous setups, upon considering a variety of performance dimensions and for networks up to 31 or 127 devices, RSS provides the best Plan Cost Improvement vs Algorithm Execution Time trade-offs ③ In heterogeneous setups, upon considering a variety of performance dimensions and more than 127 sites, the time-capped version of ESQ is the most preferable candidate ④ In homogeneous setups, upon considering a variety of performance dimensions, GREEDY is less affected by local optima and provides the best Plan Cost Improvement vs Algorithm Execution Time trade-off ⑤ In plan latency-based optimization, for networks up to 31 devices, DAG* provides the best Plan Latency vs Execution Time trade-off due to its ability to prune a considerable amount of the search space. For larger networks, the parallelism admitted by HEURISTIC and the short execution time of GREEDY overtake with respect to this trade-off. ⑥ APEROL vs competitors [7, 8], including DAG*, shows that HEURISTIC and GREEDY clearly overtake competitors, with HEURISTIC providing 3× better Plan Cost vs Algorithm Execution Time trade-off ⑦ ESC is the least dependent on shared variables and by imposing batching, it can scale up to 2.1M plans per second on commodity hardware ⑧ Experiments on the Real Testbed validate the trends observed in simulated setups, upon comparing APEROL, DAG* vs [8, 7].

5.5.5 APEROL Standalone Evaluation on Simulated Setups

Heterogeneous Setup The plots in Figures 56, 60, 58, 62 illustrate the Plan Cost Improvement of the plan devised by each APEROL algorithm, across networks of various sizes, for PRED, TRAIN, ETL and STATS workflows in the heterogeneous setup.

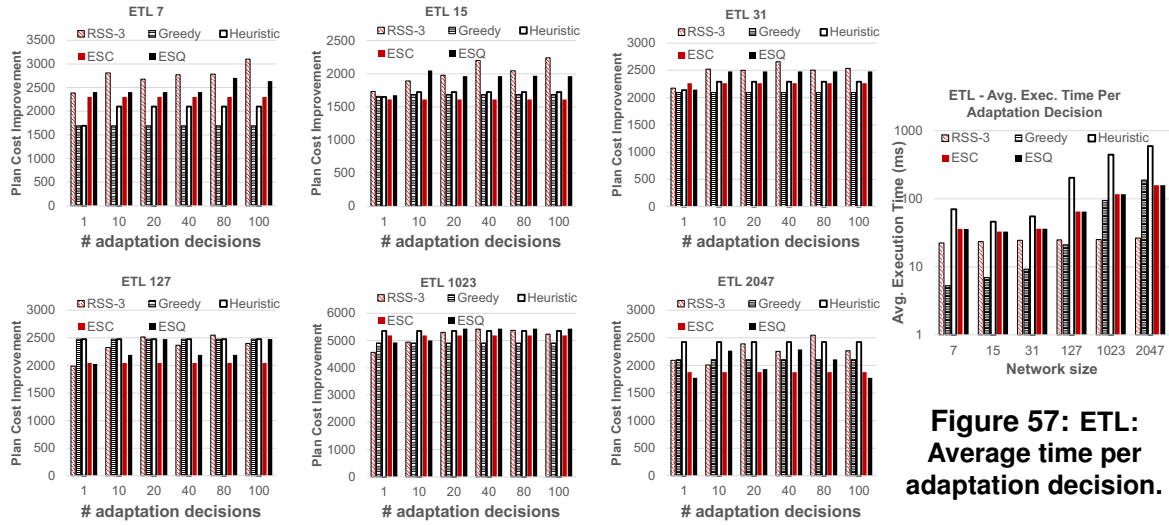


Figure 56: Heterogeneous Setup – ETL Workflow: APEROL Plan Cost Improvement (higher is better) across network sizes vs. number of adaptation decisions.

The vertical axes in the figures represent the Plan Cost Improvement of the output plan for each algorithm compared to the root plan (the higher the better). The horizontal axes represent the number of adaptation decisions within the lifetime of the workflow. In that, we provide results across networks and workflows which show conditions of various volatilities. Under rarely (respectively frequently) changing conditions, the number of adaptation decisions is low (respectively high). We couple the description of these figures, each with its own execution time chart, which shows the Average Execution Time of each algorithm per optimization decision. Hence, we can judge the plan quality vs algorithm execution time trade-offs of APEROL.

For networks up to 31 sites, RSS is the best option because it consistently provides the highest Plan Cost Improvement across various numbers of optimization decisions (Figures 58, 62, 56, 60) and simultaneously provides an average algorithm execution time of 30ms (Figures 59, 63, 57, 61). For networks of 127 and up to 1023 sites, HEURISTIC progressively overtakes RSS and for 2047 sites HEURISTIC mostly provides the best Plan Cost Improvement. However, it entails up to half an order of magnitude higher execution time compared to ESC and ESQ (recall both exhaustive algorithms have a timeout).

In the majority of the cited cases in Figures 58, ESC provides a cost improvement comparable to or worse than GREEDY. Upon an adaptation decision, ESC starts its exploration from plan

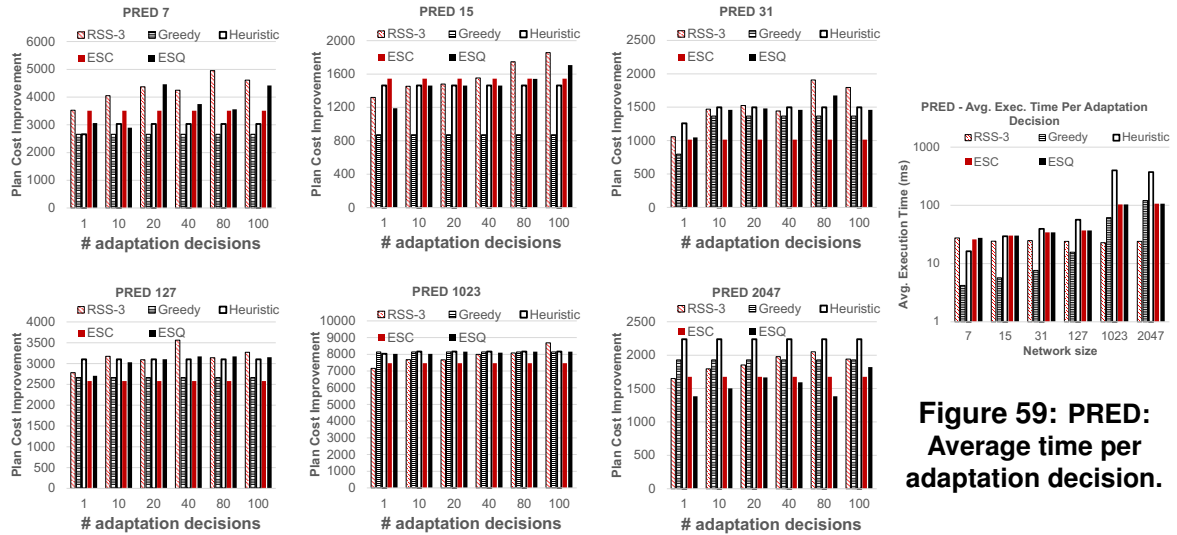


Figure 58: Heterogeneous Setup – PRED Workflows: APEROL Plan Cost Improvement (higher is better) across network sizes vs. number of adaptation decisions.

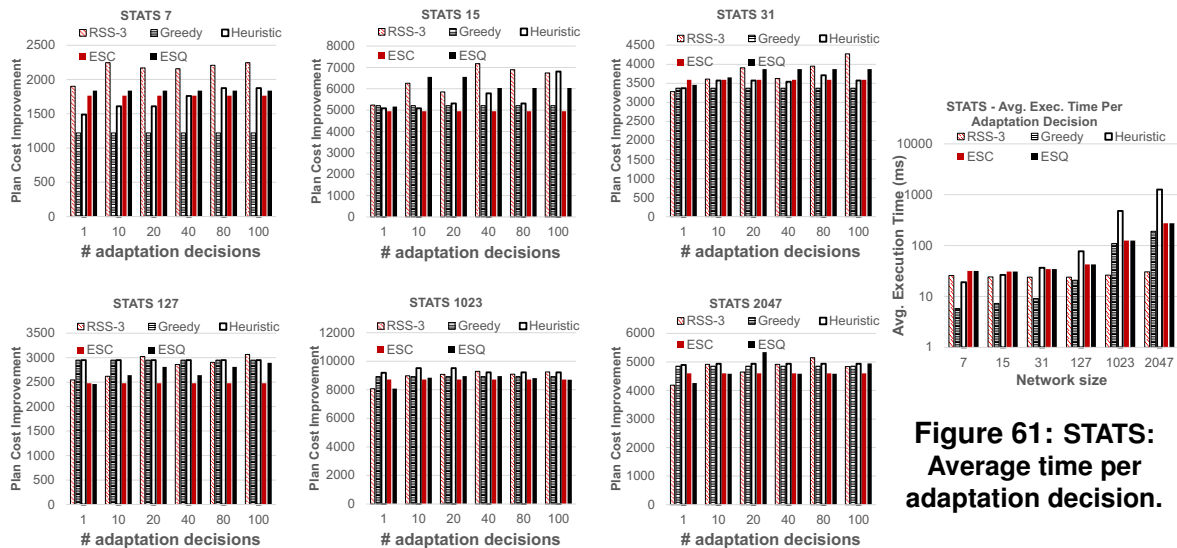


Figure 60: Heterogeneous Setup – STATS Workflow: APEROL Plan Cost Improvement (higher is better) across network sizes vs. number of adaptation decisions.

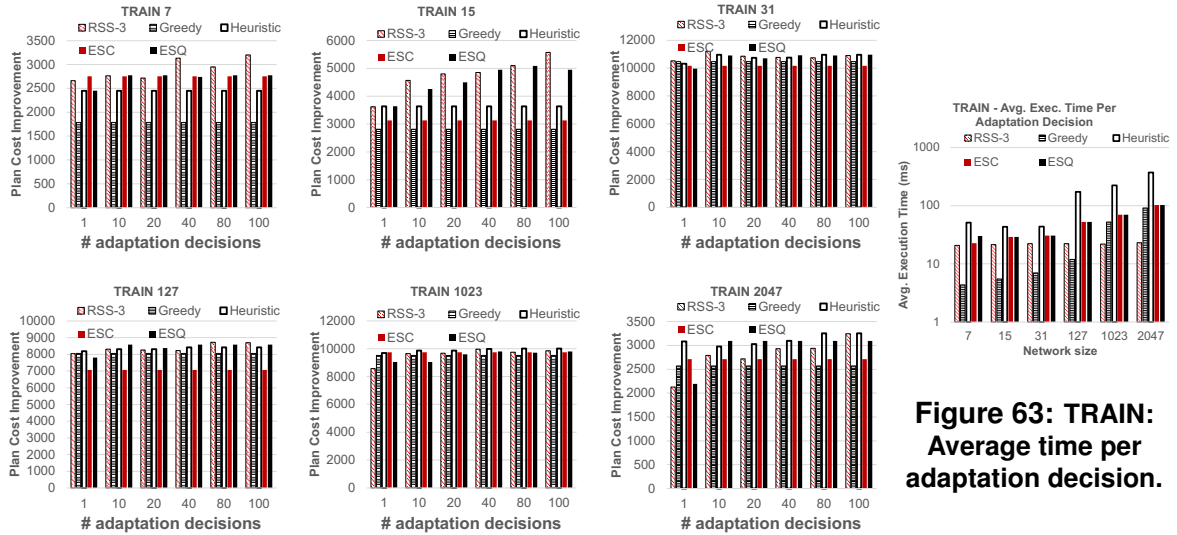


Figure 63: TRAIN: Average time per adaptation decision.

Figure 62: Heterogeneous Setup – TRAIN Workflow: APEROL Plan Cost Improvement (higher is better) across network sizes vs. number of adaptation decisions.

0. It can therefore be more detached from the current situation of the deployed execution plan. ESQ is the second best alternative across network sizes in Plan Cost Improvement and, combined with its capped execution time in Figures 59, 63, 57, 61, it is the best choice for networks of thousands of sites across the number of adaptation decisions.

GREEDY exhibits important Plan Cost Improvement between network sizes of 127 and 1023, but (a) it gives an improvement that comparable RSS and (b) it is in those networks where its execution time (Figures 59, 63, 57, 61) starts to match the execution time of RSS (for network size 127) and that of ESQ, ESC afterwards.

To sum up, in heterogeneous set ups, for small network sizes up to 31 or 127 sites, RSS provides the best Plan Cost Improvement vs Average Algorithm Execution Time trade-offs. For larger networks, ESQ constitutes the best alternative with respect to the involved trade-offs. HEURISTIC may be the best option for large networks, but only if we loosen our tolerance on the Average Execution Time of the algorithm and, thus, of the adaptation decision.

Homogeneous Setup In homogeneous setups, the previous picture is reverted. In Figure 64 and Figure 65, we plot pairs of graphs for Plan Cost Improvement, Average Algorithm Execution Time in the ETL and STATS workflows. We provide results for the middle case of 127 network size, but our observations hold across network sizes in the homogeneous setup case. HEURISTIC, GREEDY provide the best Plan Costs Improvements, while ESQ is ranked third. GREEDY simultaneously provides the lowest Average Execution Time of up to one second, making it the best choice in these setups. We re-iterate that this is a result of the homogeneity of the setting which affects less the algorithms basing their search space exploration on locally optimal decisions.

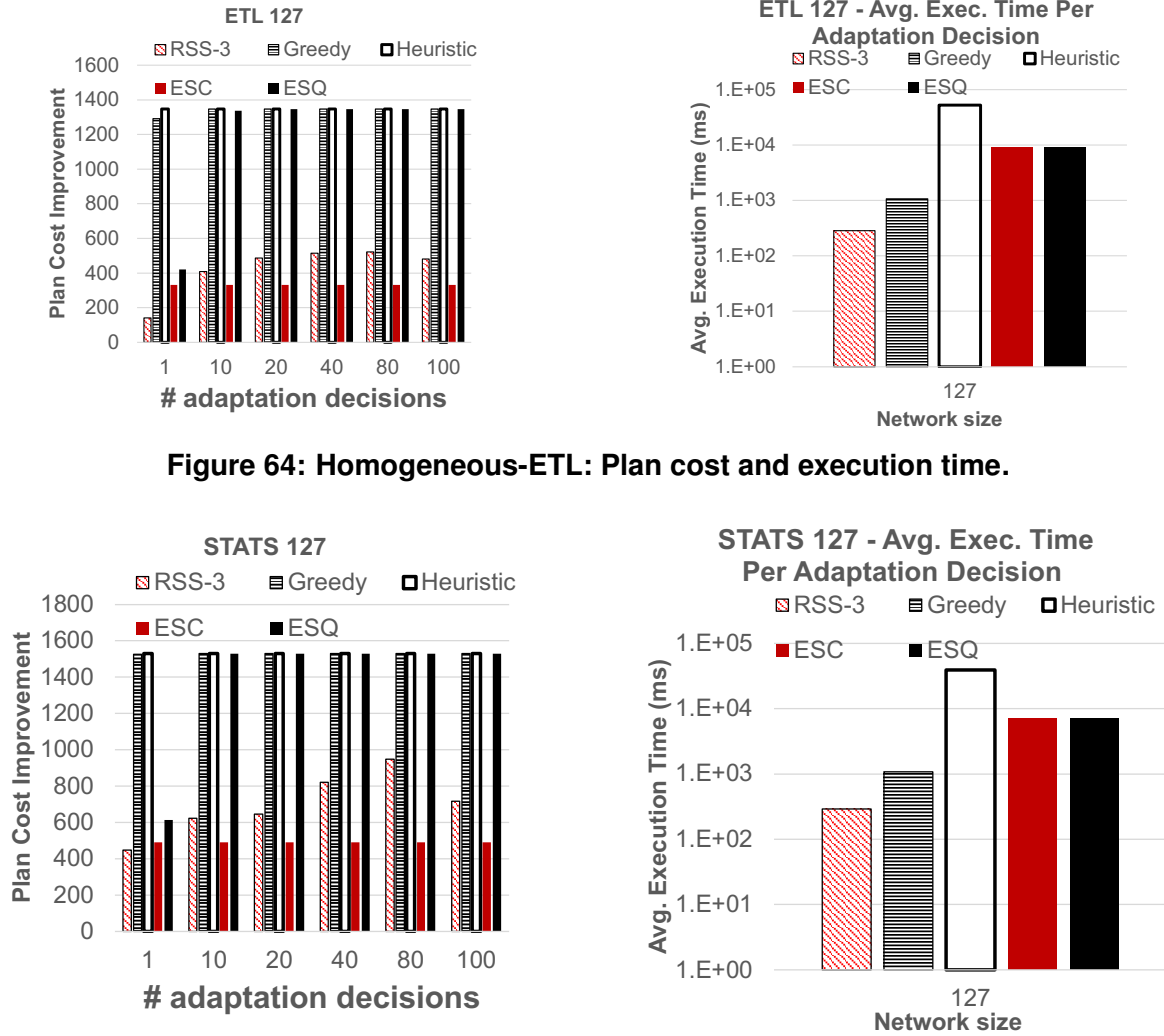


Figure 65: Homogeneous-STATS: Plan cost and execution time.

5.5.6 APEROL - DAG vs SOTA: Comparative Evaluation on Simulated Networks and Real Testbed

We now compare APEROL and DAG* [74] against Spring Relaxation (SPRING-RELAX) used in NEMO [7]; and GOVERNOR [8]. We compare these algorithms on our MacBook Air M2, both on the Real Testbed and on the simulation setups. DAG* and SPRING-RELAX operate on optimizing latency (the lower plan cost, the better for the experiments of this section) and they cannot be trivially extended to host the other performance dimensions considered in APEROL. For instance, DAG* optimality is not guaranteed for diverse performance dimensions that are non monotonic (increasing or decreasing the score as the algorithm proceeds). Similarly, SPRING-RELAX by design is based on physics and uses network latency values as "forces" to drag an operator on one or the other side of the network. Therefore, for the compared candidates and for APEROL, DAG* we consider latency as the optimization metric. The migration cost is also a latency value which accounts for the time it takes to transfer the state, upon runtime adaptation, without letting the workflow ingest and process data. Since

DAG* does not account for the migration cost upon exploring the search space, we add that migration cost to the output, optimal plan of DAG*.

In Figure 66a, Figure 66b (comparison on the simulated setup) and in Figure 66c (comparison on the Real Testbed), we show pairs of Plan Latency, Execution time for DAG*, SPRING-RELAX, GOVERNOR and GREEDY, HEURISTIC and RSS. We omit ESQ and ESC to improve readability as the trends are similar to our previous experiments.

On matched network sizes (7, 15, 31, 127) on both the Yahoo Benchmark on the Real Testbed and RIoT on the simulated setups, the qualitative ordering of algorithms and the trends are similar: HEURISTIC and GREEDY consistently yield the lowest plan latencies (Heuristic: 2.52–4.14 ms on RIoT, 1.43–4.13 ms on Yahoo; Greedy: 2.52–11.19 ms on RIoT, 1.47–7.46 ms on Yahoo). DAG* ties this front-runner pair on small graphs (2.52–4.14 ms on RIoT, 1.53–6.70 ms on Yahoo) but its execution time, while tiny on Yahoo (10–181 ms), already rises on RIoT at 127 devices (3–6,413 ms) and fails to complete beyond that scale. GOVERNOR, GREEDY offer the fastest execution but with a higher Plan Latency (Governor: 8–15 ms on RIoT, 18–43 ms on Yahoo; Greedy: 3–26 ms on RIoT, 21–53 ms on Yahoo). But GOVERNOR has a clearly higher Plan Latency (2–7× vs. HEURISTIC on RIoT; 2.31× on Yahoo). RSS maintains a nearly constant execution time (67.6–79.2 ms on RIoT; 82–105 ms on Yahoo), while its plan quality degrades as the search space grows (e.g., 5.58–19.93 ms on RIoT; 2.60–15.80 ms on Yahoo). SPRING-RELAX constitutes a worse Plan Latency–Execution Time trade-off (9.03–28.76 ms and 42–176 ms on RIoT; 5.17–7.77 ms and 54–134 ms on Yahoo). Thus, the trends of the simulated setup are mirrored in the Real Testbed.

For a more explicit ranking of the algorithms we rank them by (i) Plan Latency and (ii) Execution Time separately. We take the respective performance values of each algorithm for each (workflow, network size) point, (separately for Plan Latency and Average Execution time, respectively), averaging each algorithm's performance across networks and benchmarks. We then compute a combined score via the Harmonic Mean of Plan Latency and Execution time.

Ranking on Average Plan Latency (lower is better):

HEURISTIC (4.42 ms) < DAG* (4.76 ms, when it completes) < GREEDY (8.46 ms) < RSS (18.22 ms) < SPRING-RELAX (20.47 ms) < GOVERNOR (21.29 ms)

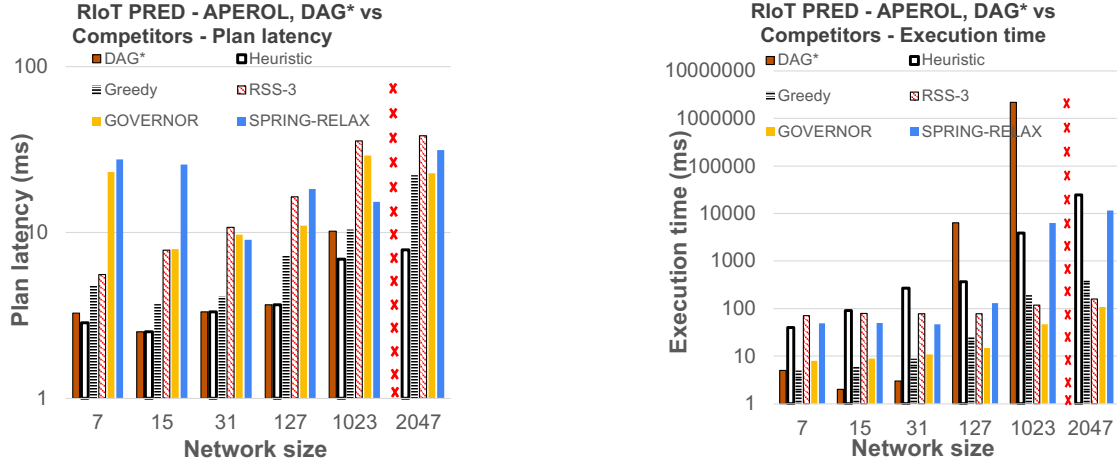
Ranking on Average Execution Time (lower is better):

GOVERNOR (29.35 ms) < GREEDY (73.47 ms) < RSS (93.16 ms) < SPRING-RELAX (2,195.24 ms) < HEURISTIC (2,640.24 ms) < DAG* (146,379.81 ms, when it completes)

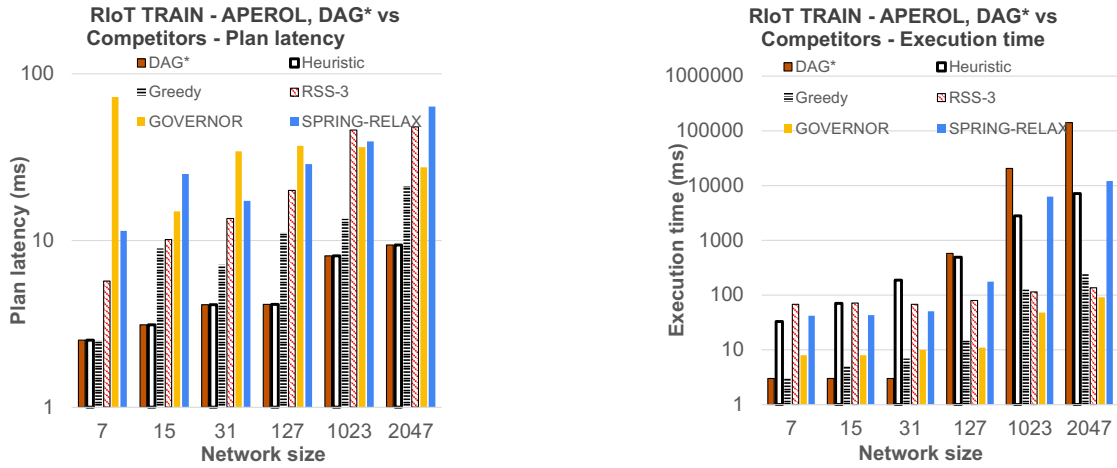
In all, DAG* achieved the best Plan Latency vs Execution Time trade-off for networks up to 31 devices. For networks of 127 or more devices HEURISTIC, GREEDY constitute the best option.

5.5.7 Scalability for Extreme Scale Analytics

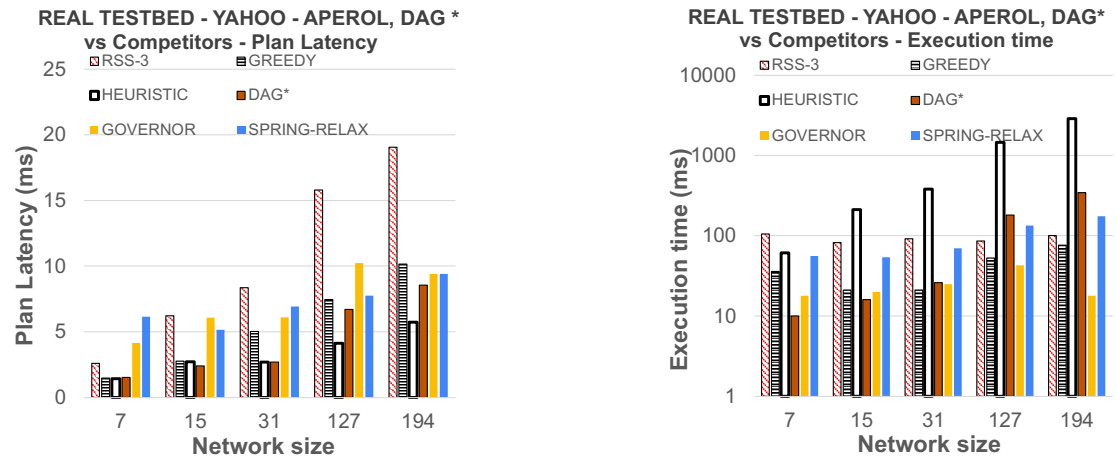
APEROL Scalability & Batching For APEROL, a way of reducing the effect of potential parallelization barriers is the introduction of batching i.e, instead of having APEROL workers examine one plan each, each worker is assigned a range of plans to examine based on APEROL's plan enumeration scheme. In Figure 67a, we study the effect of parallelism coupled with batching. Batching reduces contention among shared variables and affects the rate of examined plans. We present results for ESC, but the increasing trends are similar for all APEROL algorithms. The black lines (prefixed with MAC) in Figure 67a, plot ESC on



(a) PRED - APEROL, DAG* vs. competitors: plan latency (left, lower is better) and average execution time in a heterogeneous setup.



(b) TRAIN - APEROL, DAG* vs. competitors: plan latency (left, lower is better) and average execution time in a heterogeneous setup.

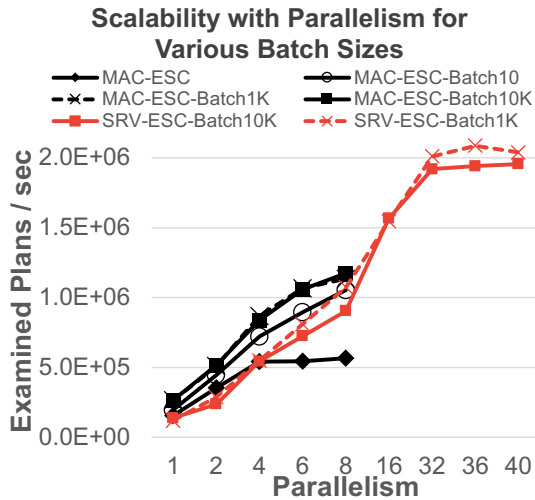


(c) YAHOO - APEROL, DAG* vs. competitors: plan latency (left, lower is better) and execution time on the real testbed.

Figure 66: APEROL, DAG* vs. competitors on synthetic (PRED, TRAIN) and real (YAHOO) workloads. Each row shows plan latency (left) and execution time (right).

the MacBook Air M2 (up to 8 cores). ESC without batching increases the rate of examined plans linearly up to a parallelism of 4, but then a plateau appears due to barriers. This is the reason we chose a default parallelism of 4 in our previous experiments. Batches of 10, 1K and 10K size, increase the rate of examined plans linearly with the full range of parallelism. ESC-BATCH10 has the major effect, almost doubling the examined plans per second for a parallelism of 8. ESC-BATCH1K improves ESC-BATCH10 by up to +100K plans per second. Further increasing the batch size to ESC-BATCH10K has little effect, which is because threads are overutilized. In Figure 67a, up to 1.2M plans/sec are examined by tuning the batch size.

We also test scalability on a server with Intel(R) Xeon(R) Silver 4310 CPU @ 2.10GHz, 20 cores/ 40 threads, and 256Gb RAM (SRV lines in Figure 67a). ESC examines more than 2M plans per second on par with batching. Hence, upon deploying APEROL on more powerful hardware, the rate of examined plans keeps scaling before reaching a plateau (on 36 – 40 threads), due to contention on shared variables. Note that for the same parallelism (e.g. 4, 8) and batch size, MAC has higher throughput due to faster CPU clock speed.



(a) Scaling with parallelism for various batches and hardware setups

	DAG*	Exhaustive
Rep#1	104893	11 Million
Rep#2	105166	11 Million
Rep#3	43818	11 Million
Rep#4	104960	11 Million
Rep#5	104898	11 Million
Average	92747	11 Million

(b) #Iterations, adversarial scenario of $\sim U(0, 10)$ costs

DAG* Pruning Capacity In the workflows presented so far, we demonstrated the execution time of DAG* without commenting on its capability of pruning the search space via the use of its admissible heuristic. In all previous scenarios, where DAG* concluded its execution, it essentially managed to prune up to 3 orders of magnitude of the search space. To better demonstrate the pruning capacity of DAG*, we here stress test the algorithm in adversarial scenarios. Adversarial scenarios involve setups where DAG* needs more iterations to find the optimal plan. The first adversarial scenario is when all nodes possess configurations with uniform cost values. In such cases, the partial graphs in the priority queue of DAG* will have similar estimated costs during the iterations of the algorithm, and many partial graphs will be expanded one after the other instead of having few promising partial graphs being prioritized for expansion. This phenomenon, in turn, delays the extraction of the optimal graph. To attest for that scenario we create a setup with the STATS workflow with 15 network devices and we assign to all configurations of all nodes uniform cost values $U(0, 10)$, fixing all edge

weights to the middle of that interval, i.e., communication latencies are fixed to 5. We repeat the experiment 5 times (Rep#1 to Rep#5) and we measure DAG* iterations. As Table 67b shows, DAG* saves 2 orders of magnitude per experiment and on average compared to the entire search space, i.e., what an Exhaustive Search will explore.

5.6 Optimization Lessons Learned

The project results confirm that optimizer selection for cloud-edge analytics must be setting-aware, and that APEROL and DAG* should be treated as complementary rather than competing “one-size-fits-all” solutions. APEROL’s key advantage is its general purpose, multi-objective formulation (throughput, latency, communication, migration), including a principled plan-switch decision that accounts for migration overhead. In contrast, DAG* adopts an A*-like, pruning-based approach that can return plans that are optimal with respect to end-to-end latency, enabled by an admissible heuristic. This makes it particularly attractive when absolute optimality is a main concern, latency is the dominant KPI and problem sizes remain small to medium for the method to complete reliably.

From an operational perspective, the evaluation highlights clear regime-dependent choices. In heterogeneous networks, the best physical plan quality vs algorithm execution time trade-off is achieved by RSS for networks up to ~31 or ~127 devices, while time-capped ESQ becomes the most preferable alternative beyond that scale. In homogeneous deployments, GREEDY becomes more reliable because local-optimum effects are less harmful, yielding strong plan improvements with low execution time. These findings also reinforce the importance of implementation-level scalability measures: for APEROL, batching reduces contention and materially increases examined-plan throughput, reaching >2M plans/s on commodity hardware. Finally, experiments on the real testbed mirror the qualitative trends seen in simulated setups, increasing confidence that the observed trade-offs and regime boundaries generalize beyond synthetic setups.

6 Text mining for Event Extraction

6.1 Introduction

This section presents the text mining components developed under Task 4.5 of the CREXDATA project. The main objective of this task is to design and implement a social media toolkit capable of crawling, extracting, and integrating information from multilingual social media sources (MO1.3).

The toolkit has been developed with a focus on weather emergency use cases, aiming to support civil protection authorities by providing an enhanced perspective on information shared by individuals affected by wildfires and floods. This capability helps improve situational awareness and address gaps that often exist in conventional information sources. While the initial focus is on weather-related events, the toolkit can be adapted to other use cases with the appropriate training of new language models using domain-specific data.

Within the weather emergency context, the primary objectives are to filter relevant social media posts in real time—with an average latency of approximately five seconds—and to extract actionable information that authorities can use to plan and evaluate rescue operations. Furthermore, question-answering models are employed to generate periodic or after-action reports, supporting the continuous improvement of future emergency response strategies.

To ensure seamless integration within the CREXDATA ecosystem, the toolkit leverages Apache Kafka for data streaming, Apache Flink for real-time stream processing, and RapidMiner AI Hub for model scoring. These components are managed through graphical workflows in RapidMiner AI Studio and a Gradio-based graphical user interface (GUI), providing users with an environment for monitoring and managing the system's functionality.

6.2 System Architecture

6.2.1 Overview

Figure 68 presents the high-level system architecture of the social media toolkit. The toolkit consists of three main components: a crawler that collects social media messages from a microblogging platform; a stream processing module that uses a language model to classify the collected posts as *fire*, *flood*, or *none* based on the weather crisis event; and a question-answering module that processes batches of relevant messages to extract valuable information. All components operate seamlessly within the CREXDATA ecosystem, utilizing the AI Hub endpoints, Docker containers, Apache Kafka clusters, and AI Studio workflows provided by the platform.

The toolkit also includes a graphical user interface (GUI) that connects with each component of the system. The interface allows users to initiate the collection of social media messages, view classified posts, and interact with the question-answering module. Several of these actions can also be performed through RapidMiner AI Studio workflows. Additionally, the toolkit supports the training of new language models for event-type prediction, which can be executed either via AI Studio workflows or dedicated Python scripts.

In a weather crisis scenario, a user triggers collection of posts by sending a JSON message

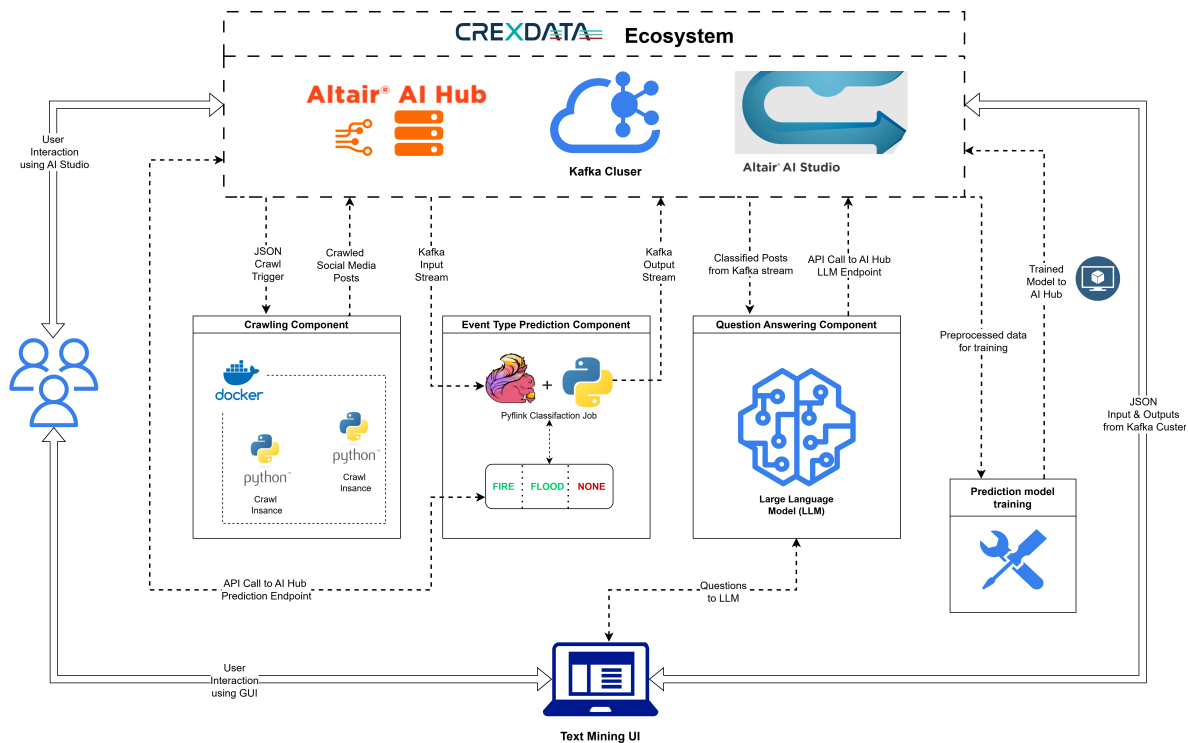


Figure 68: High-level system architecture of social media toolkit

to the CREXDATA Kafka clusters, which is received by the crawler component. The collected messages are then processed and classified by the event-type prediction component before being made available in the text mining user interface (UI). Through the UI, users can review the most relevant messages and utilize the question-answering component to extract additional insights from the collected data.

6.2.2 Development Methodology

We adopted a component-based development approach to manage the complexity of the toolkit and to apply the most suitable technologies to each part. The components were mainly developed in Python, leveraging established machine learning and data processing libraries. In particular, the event-type prediction model is based on the Transformers architecture, which enables quick and efficient classification of social media messages related to weather crisis events.

The components communicate and share data primarily through Apache Kafka, which provides high throughput for handling large-scale data streams and ensures reliability through its built-in fault tolerance. Moreover, the event-type prediction and question-answering models are deployed as API endpoints within RapidMiner AI Hub, enabling smooth integration with other system modules. Docker containerization is applied where necessary to keep components isolated, reliable, and easy to deploy. Finally, the prediction process leverages Apache Flink to ensure real-time message processing and maintain overall system fault tolerance.

The complete codebase of all components are available on GitHub ¹³

6.3 Crawler Component

A crucial aspect of extracting value from social media messages published during a weather emergency is the acquisition of relevant posts. Several social networking platforms can be used for this purpose; however, microblogging platforms are generally preferred because they focus primarily on textual content. These platforms typically provide application programming interfaces (APIs) to access and download messages, although such access is often not free. When selecting a social network, it is also important to consider the demographics of the target region to identify the most widely used platform. For example, Twitter/X might be more popular in North America, while Threads might be gaining wider adoption in Germany.

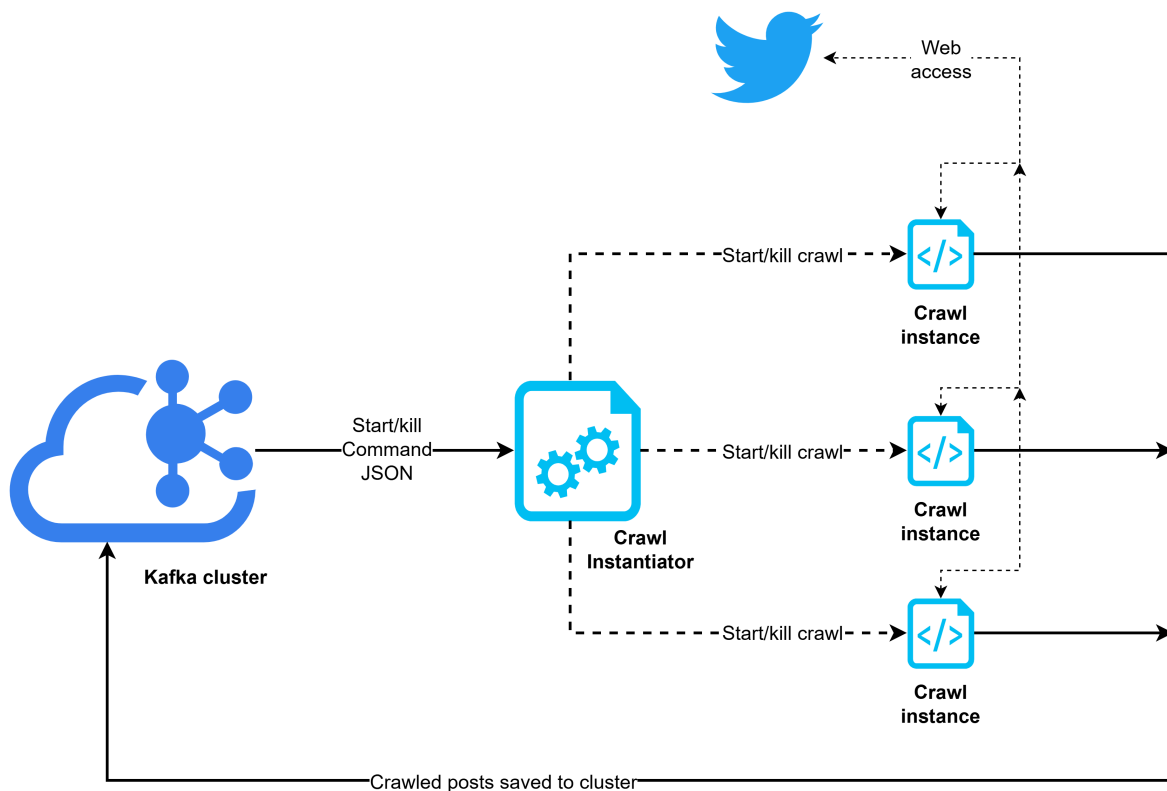


Figure 69: Crawler Component

For experimentation and proof of concept, we developed the crawler component focusing on Twitter/X. Since the Twitter API is not freely available, the crawler retrieves data directly from the Twitter web interface. This approach, while functional, is not ideal because data collected from web scraping lacks guarantees of completeness or order and may omit important information. Additionally, the HTML structure of the Twitter webpage is subject to change, which can disrupt the scraping process. The method is also slower than API-based collection, as intentional delays are introduced to avoid placing excessive load on the website. Therefore,

¹³<https://github.com/altairengineering/crexdata-public/tree/main/CREXDATA%20System%20Components/Text%20Analytics>

for operational or live scenarios, we recommend obtaining paid API access and using the official Twitter API for data collection.

Figure 69 illustrates the internal workflow of the crawler component. The component includes a crawl instantiator, implemented as a Kafka consumer that continuously listens for crawl commands from the Kafka cluster. When a user initiates a crawl—either through the text mining user interface (UI) or the AI Studio workflow (see Fig. 68)—a JSON start command is sent. This command includes the user's Twitter login credentials and parameters such as geographical coordinates (longitude, latitude), search radius, and language filters (see Fig. 70). In cases where Twitter requires two-factor authentication, the provided email credentials (i.e. *crawl_params.email*, and *crawl_params.emailpass*) allow the crawler to access the corresponding verification email. More technical details on this process are available in the project's GitHub repository ¹⁴. Each crawl command includes a unique identifier used to track and manage crawl instances. This identifier can also be used to terminate a crawl when necessary, requiring only the corresponding crawl ID.

```
--Start command--
{
  "crawl_id": "<EVENT_TYPE>_<LOCATION>_<DATE>",
  "command": "start",
  "crawl_params.username": "<USERNAME>",
  "crawl_params.password": "<PASSWORD>",
  "crawl_params.email": "<EMAIL>",
  "crawl_params.emailpass": "<EMAILPASS>",
  "crawl_params.lat": "<LAT>",
  "crawl_params.lon": "<LON>",
  "crawl_params.radius": "<RADIUS_KM>",
  "crawl_params.lang": "<LANGUAGE>"
}

--Kill command--
{
  "crawl_id": "<EVENT_TYPE>_<LOCATION>_<DATE>",
  "command": "kill",
  "crawl_params.username": null,
  "crawl_params.password": null,
  "crawl_params.email": null,
  "crawl_params.emailpass": null,
  "crawl_params.lat": null,
  "crawl_params.lon": null,
  "crawl_params.radius": null,
  "crawl_params.lang": null
}
```

Figure 70: Sample start and kill JSON crawl commands.

Once a start command with a new crawl ID is received, the crawl manager in each instance initiates a script that logs into Twitter/X and constructs a URL to search for posts within the

¹⁴<https://github.com/altairengineering/crexdata-public/tree/main/CREXDATA%20System%20Components/Text%20Analytics/Web%20Crawler%20Component/X-Crawler>

area specified in the crawl command configuration. To continuously capture the most recent posts, the webpage is refreshed every two to three seconds, and any new messages are collected and forwarded to the Kafka cluster. Figure 71 illustrates an example of a Twitter post in JSON format as sent to the Kafka cluster.

```
{
  "tweet_url": "https://x.com/testUser01/status/USER01xxxxxxxxxxxx",
  "tweet_username": "testUser01",
  "tweet_id": "USER01xxxxxxxxxxxx",
  "tweet_timestamp": "2025-11-07T11:50:34+00:00",
  "tweet_text": "It raining hard and there is a fire next door",
  "event_identifier": "fire_<LOCATION>_<DATE>"
}
```

Figure 71: Sample Twitter/X message saved to kafka cluster.

The entire crawler component is encapsulated within a Docker container, which ensures that the crawler instantiator remains continuously active to receive new crawl commands from the Kafka clusters. Containerization also allows the crawler to be deployed from any machine with minimal setup. Additionally, it is possible to run multiple instances of the component in parallel; however, care must be taken to ensure that they are properly configured to avoid interference between instances.

6.4 Event-type Prediction Component

In this subsection, we present the event type prediction models developed to determine the relevance of social media messages published during weather emergencies or crisis events. The concept of “*relevance*” is defined from the perspective of civil protection agencies and first responders, as the social media toolkit is primarily designed to support their operations. The goal is to identify messages that provide actionable information — such as reports of lives in danger, updates useful for monitoring relief efforts, tracking public sentiment or panic, and assessing the progression of the crisis.

To achieve this, we fine-tuned our prediction models accordingly. In the following sections, we describe our model design choices, including its architecture, training data, and fine-tuning methodology. We also outline the experiments conducted to enhance both prediction accuracy and latency performance, and conclude with a discussion on the model's integration into the CREXDATA platform.

6.4.1 Model Design Decisions

BERT. Bidirectional Encoder Representations from Transformers (BERT) [17] is a language representation model based on the Transformer architecture [82]. The Transformer architecture enables efficient pre-training of language models on large amounts of unlabeled data while retaining long-range dependencies in text through its self-attention mechanism, which serves as the basis for foundation models. BERT introduced the concept of masked language modeling (MLM) using bidirectional training, enabling language models to understand context from both directions within a sentence. Such models could be fine-tuned on relatively small,

task-specific datasets to achieve state-of-the-art (SOTA) performance across various natural language processing (NLP) tasks, such as text classification and question answering.

Since then, extensive research has built upon the BERT model, utilizing it for word embeddings or fine-tuning it for a wide range of downstream applications [33, 25, 94]. We selected the BERT language model for our work due to its SOTA performance in text classification, its efficiency with limited labeled data, its training effectiveness, and its robustness against overfitting.

Foundation Model. Once we settled on using BERT, the next step was to decide which foundation model to adopt as the base for our fine-tuning experiments. There are several open-weight models (i.e., models whose parameters and weights are publicly available) hosted on repositories such as Hugging Face¹⁵, trained on diverse datasets including web crawls, Wikipedia, books, and social media data.

To streamline our selection, we focused on masked language models (MLMs) that met two key criteria:

1. **Multilingual capability**, to align with our goal of processing multilingual social media messages; and
2. **Model size**, preferring models with around 300 million parameters to ensure fast prediction and efficient deployment.

We selected four models for our experiments: mGTE-MLM-base [91], DistilBERT [66], TwHIN-BERT-base [92], and XLM-T-base [6]. Among these, mGTE-MLM-base and DistilBERT were pre-trained on Wikipedia and data from books; TwHIN-BERT-base was pre-trained on Twitter data; and XLM-T-base is a fine-tuned version of XLM-RoBERTa-base [10], which was originally trained on Common Crawl¹⁶ data. We then fine-tuned these models on weather crisis social media data (details provided in later sections) and compared their performance using standard text classification evaluation metrics—namely, precision and recall. However, our primary focus was on the F1 score, which represents the harmonic mean of precision and recall, providing a balanced measure of model accuracy. The results presented in Table 14 show that all models performed well as base models for our task, with XLM-T-base achieving a slight advantage—likely due to its prior fine-tuning on Twitter data. Consequently, we selected XLM-T-base for the remainder of our experiments.

Data. To achieve reliable results in text classification, it is essential to build annotated (labeled) datasets composed of domain-specific texts that the model can learn from. This task is particularly challenging when addressing multilingual contexts. Since our focus is on weather crisis social media messages, we searched for publicly available datasets from past disaster events. The datasets we identified were primarily in English and are described in deliverable D1.2. Following initial experiments (see D4.1), we decided to augment our data by collecting additional posts directly from the Twitter/X webpage. We focused on past wildfire and flood incidents in different regions to capture multilingual content: Catalonia (Catalan), United States and United Kingdom (English), Germany and Austria (German), and Spain (Spanish). The collected data were then processed as follows:

¹⁵<https://huggingface.co/>

¹⁶<https://commoncrawl.org/>

Table 14: Comparison of multilingual foundation models

Models	# params.	Languages (F1)				Avg F1.
		de	ca	es	en	
XLM-T-base	~300M	0.768	0.598	0.529	0.773	0.667
DistilBERT	134M	0.785	0.462	0.454	0.780	0.620
mGTE-MLM-base	306M	0.770	0.535	0.520	0.790	0.654
TwHIN-BERT-base	280M	0.759	0.530	0.485	0.808	0.646

- **Anonymization.** This step involves removing personally identifiable information (PII). While social media posts are publicly visible, we removed Twitter/X usernames and email addresses to ensure privacy.
- **Denoising.** In this step, we removed non-essential or noisy elements from the text, including hashtags, URLs, and HTML character entities (e.g., &#x2013;).

The processing steps were intentionally kept minimal to preserve the unstructured nature of social media messages. To further augment our multilingual dataset, we translated the English texts into Catalan and Spanish, as these languages were underrepresented due to their low-resource nature, and the use of Twitter/X in those regions. For this translation step, we used SalamandraTA-7B¹⁷, a state-of-the-art translation model developed by the Barcelona Supercomputing Center, capable of translating between 35 languages. The statistics of our dataset are presented in Table 15. We ensured that the crisis incidents in the train and test sentences were different.

Table 15: Dataset showing train and test sentence counts per language.

Language	Train Sentences			Test Sentences		
	Fire	Flood	None	Fire	Flood	None
GERMAN	5371	6070	247354	222	304	7416
CATALAN	4782	32291	100922	520	340	10611
SPANISH	4974	34002	148357	592	239	5988
ENGLISH	9086	31196	189094	230	942	7318

Data annotation. Given the size of our dataset and the variety of languages it contains, manually annotating the text for our task would be highly challenging. Therefore, we explored the use of large language models (LLMs) for zero-shot text classification to automatically annotate the dataset for fine-tuning. Recent studies have shown that instruction-tuned LLMs can perform reasonably well in zero-shot text classification tasks [85].

The idea behind this approach is to provide the LLM with a carefully designed prompt that

¹⁷<https://huggingface.co/BSC-LT/salamandraTA-7b-instruct>

```
[Task]
Analyze the given body of text and determine whether it contains information
relevant to an ongoing fire disaster, flood disaster, or neither. The
classification should be based on the presence of specific, actionable
details or key indicators of relevance to such disasters.

[Instructions]
Examine the text for mentions of fire-related or flood-related events, such
as locations, impacts, safety advisories, or emergency responses.
Classify as "fire" if the text contains information about wildfires, house
fires, smoke hazards, evacuation orders due to fire, or related topics.
Classify as "flood" if the text discusses water inundation, flash floods,
storm surges, flood warnings, or evacuation due to flooding.
Classify as "none" if the text does not provide information relevant to
either disaster or lacks actionable disaster-related content.
Texts that express sympathy, offer prayers, or request donations should also
be classified as "none".
Only assign "flood" or "fire" if the relevance is clear; otherwise, default
to "none".

[OutputFormat]
The response should be formatted as follows:
{"classification": "[flood/fire/none]"}
Example
Input: "Evacuation orders have been issued for Riverside due to rapidly
spreading wildfire near the canyon."
Output: {"classification": "fire"}<EXAMPLES>
Classify:
Input: "<TEXT>"
```

Figure 72: Zero-shot annotation prompt.

clearly defines the task of assigning a class to a given text, while also instructing it on key contextual cues to consider during classification. Our work in [59] details our experiments on automatic annotation. The prompt used for zero-shot annotation is shown in Figure 72. We instructed the LLM to classify each given text as *fire* or *flood* if it contained actionable information related to the weather crisis, and as *none* if it lacked such information or merely expressed sympathy, offered prayers, or requested donations. Our automatic annotations were carried out using Microsoft's Phi-3.5-mini-instruct LLM¹⁸, selected for its compact size and the strong performance demonstrated by the Phi model family in our previous experiments (see [59]).

Label cleaning. Despite the strong performance of LLMs in various NLP tasks such as text classification, they remain susceptible to hallucinations and may occasionally fail to follow instructions accurately. To address this limitation and improve the quality of our annotations, we experimented with cleaning the labels produced by the LLM during zero-shot prompting. For this purpose, we used Cleanlab, an open-source framework developed by CleanlabAI, based on the concept of confident learning [58]. Confident learning is a data-centric approach that identifies labeling errors using the predicted probabilities of a machine learning model. We followed the recommended label-cleaning procedure described in the Cleanlab documentation¹⁹.

A logistic regression model was first used to generate predicted probabilities for the dataset

¹⁸<https://huggingface.co/microsoft/Phi-3.5-mini-instruct>

¹⁹<https://docs.cleanlab.ai/stable/index.html>

after zero-shot annotation. Cleanlab then assigned each data sample a label quality score within the range [0, 1] and identified samples with potential label errors. Data samples with erroneous labels, as well as those with label quality scores below 0.3, were removed from the training set. Additionally, samples labeled as *fire* or *flood* but having label scores below 0.7 were reassigned to the *none* class to further enhance data reliability.

Table 16: Comparison of fine-tuned models after label cleaning and score filtering.

Models	Languages (F1)				Avg F1.
	de	ca	es	en	
XLM-T-base (no cleaning)	0.768	0.598	0.529	0.773	0.667
XLM-T-base (with cleaning)	0.826	0.714	0.636	0.758	0.734
XLM-T-base (with cleaning and relabeling <0.7)	0.827	0.675	0.684	0.816	0.751

Table 16 presents the results of the label-cleaning process. After applying label cleaning, the model's performance improved by approximately 9.1%, and it increased by an additional 2.3% following the relabeling of low-quality labels to *none*. The labels shown in Table 15 were obtained after this label cleaning process.

Model finetuning. We adopted conventional methods for fine-tuning a BERT-based foundation model for downstream tasks. In text classification, this typically involves adding a classification head to the model and performing full-parameter fine-tuning. Alternative approaches exist to reduce training time, such as freezing the foundation model's layers and fine-tuning only the classification head, or using more advanced methods like LoRA [31], which fine-tunes a small subset of parameters while maintaining competitive performance. We explored these approaches (see D4.1) but ultimately continued with full-parameter fine-tuning, as it yielded the best performance. Our code implementation is available on GitHub²⁰. All models were fine-tuned on high-performance computing (HPC) nodes equipped with 4 NVIDIA Hopper H100 64GB GPUs and 2 Intel Xeon Platinum 8460Y 2.3GHz CPUs.

6.4.2 Improving prediction performance

Synthetic data generation (SDG). The remarkable performance of LLMs in generating coherent, human-like text—combined with their reliance on large volumes of written data for training—has led to a surge in the use of LLMs for synthetic data generation [57]. Given the scarcity of labeled data, particularly in weather crisis scenarios, synthetically generated data can be used to augment real-world datasets for fine-tuning text classification models.

To enhance our dataset and potentially improve model performance, we explored generating social media messages using highly reputable open-weight LLMs. Specifically, we employed

²⁰<https://github.com/altairengineering/crexdata-public/tree/main/CREXDATA%20System%20Components/Text%20Analytics/Text%20Analytics%20Component/Event-Type-Prediction>

the 8-bit quantized versions ²¹ ²² of Google’s Gemma3 ²³ (27B parameters) and MistralAI’s Mistral Small ²⁴ (24B parameters).

The LLMs were prompted to generate texts describing wildfire or flood incidents, as well as unrelated, random topics. The prompts were designed to capture four main categories:

- **Posts from affected persons’ perspectives with keywords.** In this case, the LLM was instructed to generate social media posts written from the perspective of individuals affected by a wildfire or flood. The prompt included guidance on tone and content, along with a list of keywords to incorporate. These keywords are shown in Figure 73.
- **Posts from government and meteorological agencies with warning alerts.** The LLM was instructed to simulate posts issued by government or meteorological agencies, particularly those providing public warnings and alerts. This helped introduce an institutional perspective into the dataset.
- **Posts unrelated to the crisis discussing random topics.** To simulate unrelated social media posts, the LLM was prompted to produce posts about various topics such as politics, celebrities, cancer, music, sports, food, lifestyle, memes, breaking news, personal updates, and tourism. These topics were chosen to represent general social media noise.
- **Posts related to the crisis but lacking useful information.** Here, the LLM was instructed to generate posts mentioning the incident but without providing actionable content—such as those requesting donations, expressing sympathy, criticizing authorities, or promoting conspiracy theories.

Table 17: Comparison of fine-tuned models with and without generated data.

Models	Languages (F1)				Avg F1.
	de	ca	es	en	
XLM-T-base (without SGD)	0.827	0.675	0.684	0.816	0.751
XLM-T-base (with SGD)	0.838	0.704	0.705	0.799	0.762

More information about the prompts can be found on GitHub ²⁵. After data generation, we applied the same label cleaning process described earlier. In total, we generated 37,250 posts: 9,357 in German, 10,392 in English, 9,148 in Spanish, and 8,353 in Catalan. We then fine-tuned a model using this synthetically generated data to augment our existing dataset. The results presented in Table 17 show a notable performance increase across all languages, except for a slight decrease in English, leading to an overall improvement of approximately 1.4%.

²¹<https://huggingface.co/unsloth/gemma-3-27b-it-GGUF>

²²<https://huggingface.co/unsloth/Mistral-Small-3.2-24B-Instruct-2506-GGUF>

²³<https://huggingface.co/google/gemma-3-27b-it>

²⁴<https://huggingface.co/mistralai/Mistral-Small-3.2-24B-Instruct-2506>

²⁵<https://github.com/altairengineering/crexdata-public/tree/main/CREXDATA%20System%20Components/Text%20Analytics>


```
{"flood": {
  "English": "power outage/failure,blocked highway/road/street,rescue boat,
    shelter,missing persons,flood,trapped,water,rain,hail,storm,
    flash flood,evacuate,evacuation",
  "Catalan": "Carretera/carrer/carretera bloquejada,tall d'energia,inundació,
    pluja torrencial,riada,cabal,canal,gota freda,DANA,
    persona desapareguda,persones desaparegudes,atrapat,refugi",
  "Spanish": "Carretera/calle/carretera bloqueada,corte de energia,inundación,
    lluvia torrencial,riada, caudal,cauce,gota fría,DANA,persona
    desaparecida,personas desaparecidas,atrapado,refugio",
  "German": "hochwasser,wasserwacht,gewässer,wassermasse,unwetter,unterwasser,
    gewitter,sturm,überflutet,überschwemmt,jahrhunderthochwasser,
    jahrhundertflut,vollgelaufene keller,heftiger regen,starkregen,
    überflutung,überschwemmung,starkregen,hagel,wasserschaden,
    unwettereinsätze,regenfälle,vermisste personen,schlammlawine,
    eingeschlossen,stromausfall,unwetterkatastrophe,sturzflut,
    windböen,hochwasserwarnung"},
"wildfire": {
  "English": "missing persons,damage,fire,smoke,mudslides,mudflow,
    debris flow,forest,wind,tree,firefighters,rescue,
    burning,drought,forest,park,flames,woods,emergency,
    evacuate,evacuation",
  "Catalan": "incendi,flames,foc,flama,arbre,bosc,forestal,vent,
    sequera,matollar,crema,cremar,cremant,bomber,ventós,
    tramuntana,parc,militar,emergències,colades de fang,
    esllavissades,persona desapareguda,persones desaparegudes,
    fumarada,fum",
  "Spanish": "incendio,llamas,fuego,llama,árbol,bosque,forestal,viento,
    sequía,matorral,quema,quemar,quemando,bombero,ventoso,
    parque,militar,emergencias,coladas de barro,deslizamiento,
    persona desaparecida,personas desaparecidas,humareda,humo",
  "German": "waldbrand,flurbrand,flamme,rauch,vegetationsbrand,waldfläche,
    brandgebiet,brandfläche,ausbreitung,ausgebreitet,heiß,gefahr,
    problem,probleme,problematisch,wind,hitze,waldbrandfläche,
    waldbrandgebiet,wüten,außer kontrolle,unter kontrolle,
    evakuierung,evakuiert,evakuieren,sicherheit,brennen,
    brandbekämpfung,feuerwehrmänner,flächenbrand,schadenslage,
    großschadenslage,unmöglich,waldbrände,warnung,warnen,rauchsäule,
    rauchsäulen,vernichtet,alarmierung,alarmiert,ernstfall,waldstück,
    betroffen,bäume,vermisste personen,"}}
```

Figure 73: Keywords for synthetic data generation.

Model distillation. To enhance the efficiency of our model, we explored knowledge distillation — a technique for compressing a model by training a smaller “*student*” model to replicate the performance of a larger “*teacher*” model. In our case, we followed the procedure outlined in [66]. Specifically, we reduced the number of hidden layers in a new XLM-T-base model by half (from 12 to 6), then initialized the student model by copying the even-numbered layers from our best-performing fine-tuned model (the teacher). Finally, we fine-tuned the student model using both the ground-truth labels from the training dataset and the teacher model’s output probabilities.

Table 18: Comparison of fine-tuned model and distilled model.

Models	Languages (F1)				Avg F1.	Latency (ms)
	de	ca	es	en		
XLM-T-base (fine-tuned)	0.838	0.704	0.705	0.799	0.762	4.112
XLM-T-base (distilled)	0.850	0.704	0.692	0.794	0.760	2.641

We performed the distillation process using our model augmented with synthetic data as the teacher. The results in Table 18 indicate a slight decrease in overall prediction performance as a result of the distillation. However, this comes with a substantial benefit, as the inference time is significantly reduced, nearly cutting the latency of the teacher model in half.

6.4.3 Integration

Stream processing. As shown in Figure 68, the event type prediction component is integrated into the CREXDATA ecosystem through Apache Flink (PyFlink). This setup enables the stream of social media messages collected by the crawler component to be processed in real time. To keep the Flink job lightweight, our BERT-based model is deployed as an API endpoint on the RapidMiner AI Hub. When a new message arrives, Flink calls the API endpoint and sends the text for classification. The endpoint then uses the model to score the text and returns the prediction result.

To prevent the Flink job from stalling while waiting for responses, we configured it to make up to ten asynchronous calls to the endpoint simultaneously, ensuring low-latency stream processing. As shown in Figure 74, the model’s processing time averages around six seconds, and we maintained an overall system latency between seven and ten seconds. This performance can be further improved by employing our distilled model.

Training workflow. Our models can be trained either using Python scripts or through a RapidMiner workflow. RapidMiner offers a GenAI extension that enables users to download foundation models, fine-tune them, and validate their performance directly within its graphical interface. The process only requires configuring the training parameters and preparing the training data as needed. Figure 75 shows an example of a workflow created for BERT model training.

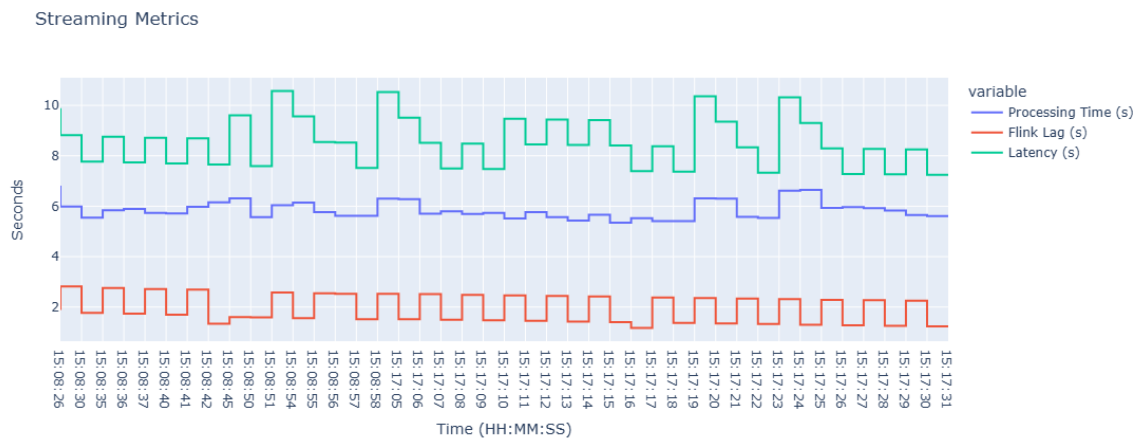


Figure 74: Stream processing latency charts.

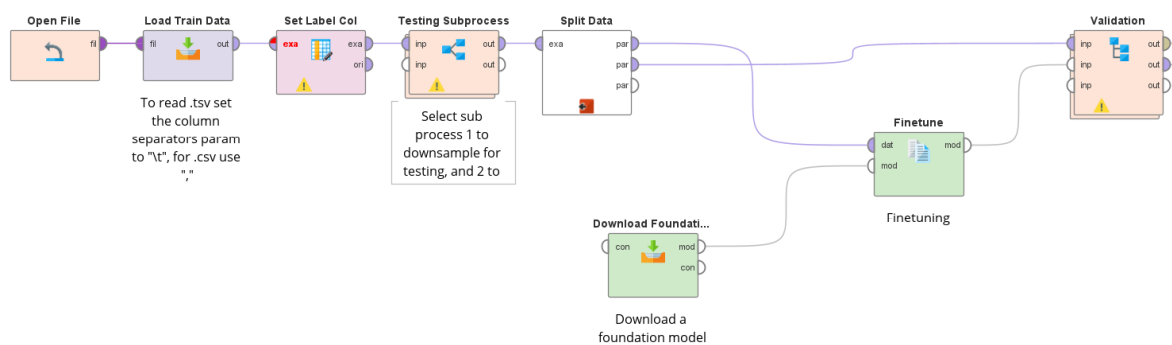


Figure 75: RapidMiner workflow for training BERT models.

6.5 Question Answering Component

In this section, we present the advancements made to the Question Answering (QA) component presented in deliverable D4.1. The QA component is designed to extract crucial information from text data by enabling users to ask focused questions about the unfolding event, and improve information access and communication during critical times, empowering informed decision-making. Here, we provide updates specifically focusing on extended evaluation, post-retrieval clustering, and integration with the overall architecture of the CREXDATA ecosystem.

The core methodology of the Question Answering component, initially documented in deliverable D4.1, was submitted to and accepted by the conference EMNLP 2024 (The 2024 Conference on Empirical Methods in Natural Language Processing). The published work includes extended evaluation and further experiments beyond the initial deliverable, validating the novelty and state-of-the-art performance of our approach. The published work can be found at [36].

6.5.1 Extended Evaluation

In D4.1, we described the core methodology of the QA component, called GENRA. In order to assess the importance of different features of the proposed approach, we ran a set of experiments on the benchmark datasets DL-19 and DL-20 [11].

In particular, we investigated the impact of passage generation, relevance assessment, and rank aggregation strategies within the GENRA framework. We use BM25 and Contriever as zero-shot lexicon-based and dense retrieval methods, respectively, for the retrieval steps within GENRA. Regarding the choice of LLMs, we opted for open-source ones (Solar and Mistral).

Passage Generation: First, we conducted experiments varying the number of passages generated per query (1, 2, 5, 10, 20). Each passage set underwent the same encoding and rank aggregation process within GENRA. We then evaluated the retrieved documents on the test data using nDCG@10. Increasing the number of passages per query initially improved retrieval effectiveness (nDCG@10), reflecting gains from information diversity. However, performance plateaued beyond 10 passages for DL-19 and 5 passages for DL-20, consistent with prior findings that excessive redundancy can hinder retrieval. Determining the optimal number of passages depends on the specific information retrieval context and the desired balance between accuracy and efficiency.

Relevance Assessment: Next, we assessed how the number m of documents selected at the relevance assessment step, influences the overall retrieval effectiveness. Varying the number of documents selected for relevance assessment revealed that effectiveness improves up to 5 documents, after which performance gains diminish. While verified passages can enhance trust and potentially improve relevance, incorporating too many can expose the method to possible misjudgments made by the LLM, leading to a decline in performance.

Rank Aggregation: Furthermore, we investigated the impact of different rank aggregation methods on GENRA's retrieval performance. We compared multiple aggregation methods (Linear, Borda, Outrank, DIBRA) against a baseline without aggregation (w/oRA). The Linear method's effectiveness might be attributed to the fact that it uses the actual ranking similarity

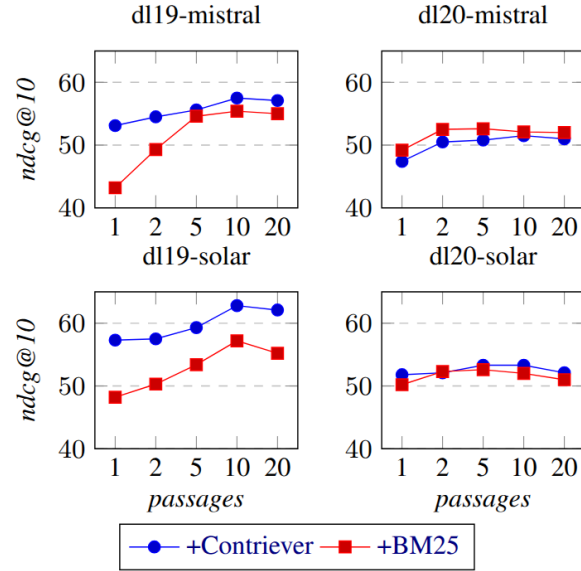


Figure 76: Impact of the number of passages.

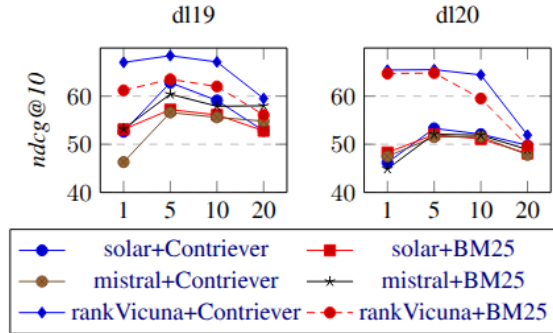


Figure 77: Impact of the number of relevant documents.

scores, in contrast to positional methods, such as Borda and Outrank. Furthermore, the more sophisticated DIBRA method might require more careful tuning for each dataset, in order to achieve optimal results. Exhaustive parameter tuning of each rank aggregation method could have produced different results, but it would hurt the generality of the GENRA method. Overall, the effect of the linear rank aggregation seems positive, improving the results obtained without it.

Efficiency: Beyond effectiveness, it is crucial to consider the computational cost associated with the use of multiple retrieval steps. Each step requires resources and can impose additional time constraints on the retrieval process. As expected, HyDE [24] has the lowest processing time across both datasets, making it suitable for scenarios where efficiency is important. On the other hand, GENRA combined with relevance judgements is slower than HyDE but faster than RankVicuna [61]. This suggests that GENRA+judgements might be a reasonable compromise between efficiency and potential effectiveness for some retrieval tasks.

GENRA	DL19		DL20	
	MAP	nDCG	MAP	nDCG
+Judgements				
+w/oRA	35.2 (42.0)	56.1 (62.2)	33.6 (33.2)	50.5 (52.9)
+borda	32.9 (36.5)	49.5 (52.8)	30.3 (25.7)	44.4 (39.7)
+dibra	32.3 (37.1)	49.7 (53.1)	26.5 (24.8)	39.9 (36.3)
+outrank	33.8 (37.6)	52.2 (53.9)	30.9 (27.3)	45.6 (42.2)
+linear	35.5 (42.3)	57.2 (62.8)	34.2 (34.7)	52.0 (53.3)
+RankVicuna				
+w/oRA	32.9 (32.5)	67.1 (67.2)	38.0 (35.3)	65.5 (63.2)
+borda	32.2 (33.7)	55.4 (61.1)	31.2 (34.1)	50.6 (55.1)
+dibra	28.3 (30.7)	52.9 (58.5)	26.4 (29.3)	46.4 (51.3)
+outrank	33.9 (34.1)	56.8 (61.4)	32.9 (34.8)	53.4 (57.1)
+linear	35.7 (36.4)	63.4 (68.4)	38.0 (39.6)	64.8 (65.5)

Figure 78: Evaluation of different rank aggregation methods within the QA pipeline. The scores represent the results of each model using BM25 for retrieval (scores in parentheses correspond to the results using Contriever).

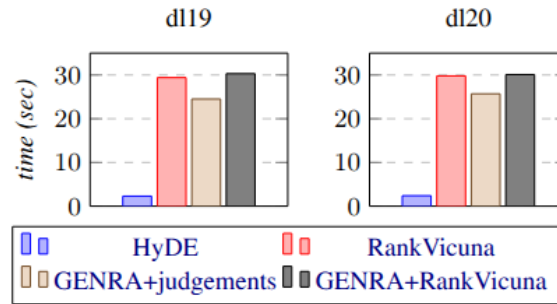


Figure 79: Average time (seconds) required for processing a query.

This ablation study provided an optimized configuration for the QA component.

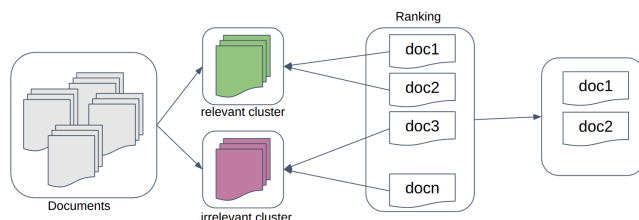
6.5.2 Post-Retrieval Clustering

In this section, we investigate the use of unsupervised clustering techniques as a post-retrieval denoising mechanism for the outputs of the QA component. The primary objective is not simply to refine the final ranking, but rather to filter out low-quality, irrelevant documents that hurt the overall retrieval performance (Figure 80a).

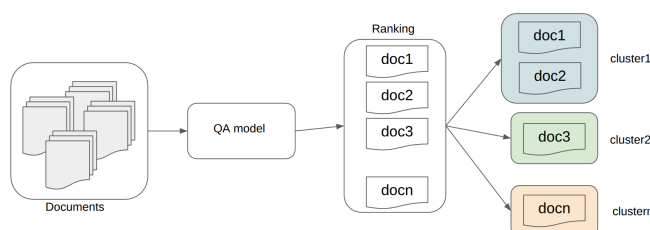
The idea is to cluster the top retrieved candidates to separate the semantically relevant documents from irrelevant or outliers. By filtering documents associated with clusters characterized as irrelevant, the method aims to improve precision and effectiveness of the final ranking.

We implemented two methodologies for filtering the top retrieved candidates via clustering:

- **Methodology 1: Direct Embedding Clustering.** This approach applies density-based clustering algorithms (e.g., DBSCAN) directly to the raw Language Model (LM) embeddings. The primary goal is to leverage the density of the vector space to explicitly identify



(a) Using clustering as a denoising mechanism.



(b) Repurposing of the generated clusters.

Figure 80: Post-retrieval clustering.

and remove noise or outlier points. Cluster relevance is determined by calculating the proximity of candidate documents to the established cluster centroids. Documents that fall outside a predefined proximity threshold are classified as noise and, thus, discarded.

- **Methodology 2: LLM-based Semantic Clustering.** This approach instructs a Large Language Model (LLM) to generate semantic labels (clusters) for documents. Documents that fall in the irrelevant clusters are then filtered out.

We assessed the performance of the proposed methodologies on two standard zero-shot retrieval benchmarks, *nfc* corpus and *scifact* [77].

The results, shown in Table 19, indicate that both clustering methodologies led to a decrease in ranking performance (NDCG@10) compared to the Baseline (the output of the QA model). Methodology 1 showed a moderate but statistically evident performance drop across both datasets. This suggests the density-based filtering (DBSCAN) was overly aggressive, thus, filtering out relevant documents that were highly ranked by the QA model. Methodology 2 led to a severe and statistically significant decline in performance. This can be attributed to errors in the LLM-based labeling.

Table 19: Post-retrieval clustering results.

ndcg@10	nfc	scifact
Baseline	0.256	0.504
Methodology 1	0.234	0.457
Methodology 2	0.043	0.188

The evaluation showed that using both clustering methodologies as a post-retrieval filtering step did not achieve the expected gains in ranking quality. Based on this finding, the clustering component could be repurposed to prioritize explainability rather than performance enhancement. Cluster labels generated through Methodology 1 could be provided to the user interface, where they could serve as additional context. For a given query, the user could identify semantically similar groups and quickly assess the relevance and thematic scope of the information presented (Figure 80b).

6.5.3 Integration

The QA component is integrated into the CREXDATA ecosystem as an API endpoint on the RapidMiner AI Hub. This endpoint is invoked by the text mining UI to perform queries over the relevant social media messages, as described in Section 6.6. Figure 81 shows the configuration of this endpoint in RapidMiner, where users can set the LLM generation parameters, including temperature, top-k, top-p, min-p, max-tokens, and repetition penalty. The endpoint currently uses a quantized (GGUF) version of Meta's latest Llama 3.2 (3B) model²⁶, which offers an excellent balance between performance and efficiency. This model can be replaced or upgraded with any other quantized (GGUF) model compatible with llama.cpp.

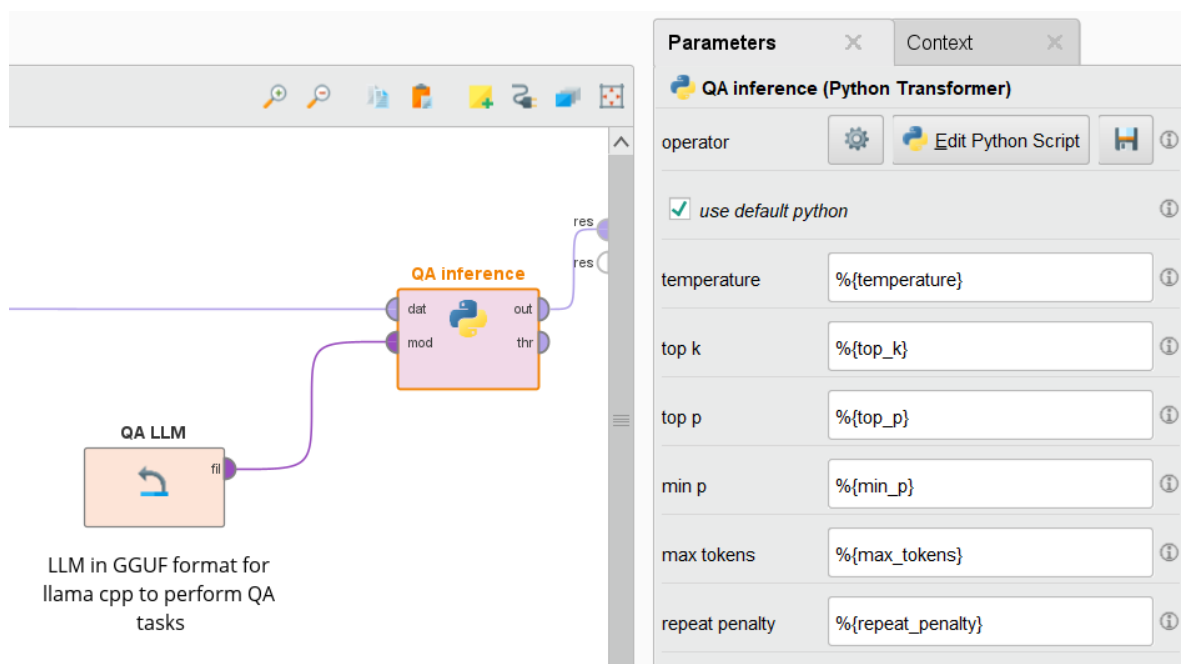


Figure 81: Question answering component as RapidMiner endpoint.

6.6 Graphical User Interface

To integrate all the tools and components of the social media toolkit, we developed a graphical user interface (GUI) that allows users to interact with the system and visualize the results

²⁶[urlhttps://huggingface.co/meta-llama/Llama-3.2-3B](https://huggingface.co/meta-llama/Llama-3.2-3B)

produced by its components. Through the interface, users can view social media posts classified as relevant and access images or links embedded within those posts.

The text mining user interface (UI) is implemented as a Gradio-based web application consisting of four main tabs—three corresponding to the core components and a fourth dedicated to debugging and logging. In addition, a sidebar is provided for initial configuration of the Kafka server and topic settings before using the application.

6.6.1 Side Bar

Mandatory Kafka Setup ⚠️

Enter kafka server details and click connect kafka

Choose server
crexdata-unauth

Bootstrap server

kafka topic with relevant tweets
text_mining_processed_relevant_tweets_stream

kafka topic sending to ARGOS
text_mining_to_argos_stream

kafka topic with relevant tweets for QA
text_mining_processed_relevant_tweets_stream

kafka topic crawler commands
text_mining_crawler_commands

kafka topic with crawler logs
text_mining_crawler_logs

Connect to kafka

Start Crawl Instance ▶

Crawl Identifier
Unique identifier to manage the crawl. Ex: flood_dortmund_20250611

X password for crawling
account password

X username for crawling
account username

Gmail app access password
Should be the app pass for the x email, which should be gmail

X email for crawling
account email

Crawl language
Only those supported by X

Start Crawl

Kill Crawl Instance □

Crawl Identifier to kill
Ex: flood_dortmund_20250611. Ensure the identifier is of a running crawl

Use via API • Built with Gradio • Settings

Figure 82: Text mining GUI: Side bar

The sidebar contains the main configuration settings of the application. Here, you can specify the CREXDATA Kafka cluster configuration, including the bootstrap server and the Kafka topics required by each of the text mining component tabs. The available configuration options are as follows:

- **Choose server:** A dropdown menu that allows you to select between two options — *crexdata-auth* and *crexdata-unauth*. The *crexdata-auth* option connects to a pre-configured Kafka cluster that requires authorized access using the *SASL_SSL PLAIN* security protocol, while *crexdata-unauth* is for open servers without authentication (e.g., *localhost:9092*). More information on configuring these options can be found in the GitHub repository ²⁷.

²⁷ <https://github.com/altairengineering/crexdata-public/tree/main/CREXDATA%20System%20Components/Text%20Analytics>

- **Bootstrap server:** This text box allows you to provide the bootstrap server address (e.g., *localhost:9092*). It is available only when the *crexdata-unauth* option is selected in the Choose server dropdown.
- **Kafka topic with relevant tweets:** As the name suggests, this option sets the Kafka topic containing social media messages classified as relevant by the event-type prediction component. Note that this topic must match the corresponding topic configured for this purpose in the event-type prediction component, and the topic must already exist.
- **kafka topic sending to ARGOS:** Defines the Kafka topic monitored by the ARGOS system for visualizing simulations, posts, and other data on the map. This topic should already exist and be properly configured in ARGOS. Further details about ARGOS can be found in deliverable D2.2.
- **kafka topic with relevant tweets for QA:** Specifies the Kafka topic containing relevant social media messages used by the question-answering component to extract information. This topic can be the same as the *Kafka topic with relevant tweets*.
- **kafka topic crawler commands:** Sets the Kafka topic used to send start crawl and kill crawl commands to the crawler component.
- **kafka topic with crawler logs:** Defines the Kafka topic for crawler logs, used for debugging and for tracking active or terminated crawl instances. This topic should also be configured within the crawler component.

A sample configuration is shown in Figure 82.

6.6.2 Crawler UI Tab

The screenshot displays the CREXDATA Crawler UI tab. On the left, a sidebar contains the 'Mandatory Kafka Setup' section with fields for 'Choose server' (set to 'crexdata-unauth'), 'Bootstrap server', and four Kafka topics: 'kafka topic with relevant tweets' (text_mining_processed_relevant_tweets_stream), 'kafka topic sending to ARGOS' (text_mining_to_argos_stream), 'kafka topic with relevant tweets for QA' (text_mining_processed_relevant_tweets_stream), and 'kafka topic crawler commands'. The main panel has tabs for 'Crawler UI', 'Relevance UI', 'Question Answering UI', and 'DEBUG/LOG Console'. The 'Start Crawl Instance' section includes input fields for 'Crawl Identifier' (Unique identifier to manage the crawl. Ex: flood_dortmund_20250611), 'X password for crawling' (account password), 'X username for crawling' (account username, set to 'TCrexdata'), 'Gmail app access password' (Should be the app pass for the x email, which should be gmail), 'X email for crawling' (account email, set to 'textmining.crexdata@gmail.com'), 'Crawl language' (Only those supported by X, set to 'German'), 'Lat of location to crawl from' (in Decimal, set to '51.513889'), 'Lon of location to crawl from' (in Decimal, set to '7.465278'), and 'Radius to crawl from' (in Km, set to '100'). A 'Start Crawl' button is at the bottom of this section. The 'Kill Crawl Instance' section has a 'Crawl Identifier to kill' field (Ex: flood_dortmund_20250611. Ensure the identifier is of a running crawl) and a 'Kill Crawl' button. At the bottom right, there are links for 'Use via API', 'Built with Gradio', and 'Settings'.

Figure 83: Text mining GUI: Crawl UI tab

The Crawler UI tab is designed specifically for starting and terminating crawl instances. It includes input fields for providing the configuration details required for a crawl, as described in Section 6.3. To use the crawler component through this tab, users need to fill in the required

inputs, including Twitter/X login credentials, crawl parameters, and a unique crawl identifier, then click *Start Crawl*. To stop a running crawl, the user simply provides the corresponding crawl identifier and clicks *Kill Crawl*. The layout and contents of this tab are shown in Figure 83.

6.6.3 Relevance UI Tab

The screenshot shows the 'Relevance UI' tab with the following components:

- Kafka polling parameters:**
 - Number of messages to return: 10
 - Poll Timeout: 6 seconds
- Get relevant posts:** A button to refresh the tweet list.
- Number of Tweets Reviewed:** 10
- Select tweet index to send to ARGOS:** A row of checkboxes for indices 0 through 9. Indices 2 and 6 are selected.
- Save to ARGOS:** A large green button to send selected tweets to the ARGOS system.
- Select a tweet_id in the table to view Embedded tweet:** A label for the tweet table.
- Tweet Table:**

index	tweet_text	tweet_url	tweet_id	tweet_timestamp	tweet_lat	tweet_lon	event_identifier	event_class	event_score
6	Severe #Storms leading to #Flooding in Tirol: mambolook.com/natural-disast...	https://twitter.com/ScienceTopNews/status/749398016879747972	749398016879747972	2025-05-26T11:12:00+00:00	47.26484616637628	11.391587476473843	flood_testset	flood	0.9960438013076782
	Retweeted RiskMap (@Riskmapintel): Severe Storms Leading To	https://twitter.com/MLM_PA	74935522	2025-05-					
- Embedded Tweet:** A preview of the tweet from Science News (@ScienceTopNews) showing a photo of a car in floodwaters.

Figure 84: Text mining GUI: Relevance UI tab

The Relevance UI tab is designed to display social media posts classified as relevant by the event-type prediction component. Additionally, this tab allows users to send selected posts to the ARGOS system for visualization on the map. To retrieve messages from the Kafka cluster, the tab requires two parameters:

- **Number of messages to return:** Specifies the number of messages to poll from the Kafka topic containing relevant posts. This limits the number of posts displayed on the webpage at a time. Each time the *Get relevant posts* button is clicked, a new batch of posts of the specified size is retrieved.
- **Poll timeout:** Defines the number of seconds to wait when retrieving messages from the Kafka topic. A value between 4-6 seconds is recommended.

In a live scenario, after setting these parameters, the user clicks *Get relevant posts* to load the messages. In the input field for selecting posts to send to ARGOS, users specify the posts by their index number as shown in the displayed table. To view an embedded post, users can click its corresponding *tweet_id*. Finally, clicking *Save to ARGOS* sends the selected posts to the ARGOS system. To fetch a new set of relevant posts, users can click *Get Relevant Posts* again. The layout and contents of this tab are shown in Figure 84.

6.6.4 Question Answering UI Tab

Crawler UI Relevance UI **Question Answering UI** DEBUG/LOG Console

Kafka polling parameters

Event ID
unique event id used for crawling

Number of recent tweets to query
Round value
50

Get recent posts

Choose QA Type
no-RAG is fast, while Genra-RAG is slow but more accurate
no-RAG

Choose Response Language
the language to get your response
English

Choose Query Type
choose query to ask a question, or summarise to get a summary of the messages.
query

Ask a question about the incident.

Examples

Which streets and locations are mentioned most frequently? W... Is there evidence of vulnerable infrastructure that could be... Where are reports of impassable streets or underpasses occur... Which posts indicate an acute danger to human life?

Where do people report water in their living areas, not just... Do you find evidence of trapped people (e.g., underground pa...

Run QA

Response.

Figure 85: Text mining GUI: Question Answering tab

The Question Answering (QA) UI tab provides an interface for users to query the relevant posts using a large language model (LLM). It enables information extraction through the question-answering component described in Section 6.5. The process requires the following parameters:

- **Event ID:** Used to filter relevant posts retrieved from the Kafka topic for question answering. This corresponds to the crawl ID (see Section 6.3) used to identify the event for which the posts were collected. This filter is essential when multiple events are saved to the same Kafka topics.
- **Number of recent tweets to query:** Specifies the number of messages to poll from the Kafka topic containing relevant posts that will be analyzed by the LLM. By default, this value is set to 50, but it can be increased to include more posts in the question-answering process. However, larger values will increase processing time.
- **Choose QA Type:** A dropdown menu for selecting the type of question-answering process. Two options are available: *Genra-RAG* (with retrieval augmented generation) and *no-RAG* (without RAG). The *Genra-RAG* option provides higher accuracy and returns the top five posts most relevant to the user's query, but it is significantly slower than *no-RAG*.
- **Choose Response Language:** A dropdown menu to select the language in which the LLM should respond. The four supported languages are English, Spanish, German, and Catalan.
- **Choose Query Type:** A dropdown menu to define the type of request sent to the LLM. The *query* option allows the user to enter a custom question, while the *summarise* option generates a summary of the incident described in the relevant posts.
- **Ask a question about the incident:** An input field where the user can type a question

for the LLM. This field is only active when the Query Type is set to *query*.

> Run QA

Response.

Based on the provided documents, it appears that the streets and locations mentioned most frequently are:

1. Innsbruck (#Innsbruck, 7 times)
2. Tirol (#Tirol, 3 times)
3. Amras (#Amras, 4 times)
4. DEZ (Durchgangsstraße, 5 times)

As for the situation, it seems that there has been severe weather and flooding in Innsbruck and surrounding areas, particularly in Amras. Documents #2, #4, #6, #10, #11, #12, #13, #14, and #15 all mention flooding, strong rain, and water damage.

The situation is likely chaotic, with people experiencing disruptions to their daily lives due to the floodings. Some documents (#3, #7, #8) also suggest that there have been severe storms leading to flooding, which may indicate a larger-scale weather event.

It's worth noting that some documents (#9, #16) provide updates on the situation after the initial flooding, with reports of ongoing efforts by the fire department and residents dealing with the aftermath.

Select a tweet_id in the table to view Embedded tweet



index	tweet_text	tweet_url	tweet_id	tweet_timestamp	tweet_lat	tweet_lon	event_identifier	event_class	event_score
7	Severe #Storms leading to #Flooding in Tirol: mambolook.com/natural-disast...	https://twitter.com/ScienceTopNews/status/749390816879747072	749390816879747072	2025-05-26T11:12:00+08:00	47.26484616637628	11.391587476473843	flood_testset	flood	0.9968438813876782
	Retweeted								

Figure 86: Sample no-RAG response in question answering process

Figure 85 shows the Question Answering (QA) UI tab. Once all parameters are properly configured, the user can click *Get recent posts* to retrieve posts for processing and then click *Run QA* to generate a response from the LLM. When the *no-RAG* option is selected, the output is a simple textual response, as illustrated in Figure 86. In contrast, when the *Genra-RAG* option is used, the output is a JSON object, as shown in Figure 87.

The JSON object includes a batch number, since the process runs in batches to reduce processing time. It also contains the query, the LLM-generated response, and the top five posts identified as the most relevant answers to the query. Each post entry includes an index (to locate it in the displayed table), a relevance score, and the textual content of the message.

```
[{
  "batch": 0,
  "question": <QUERY>,
  "summary": <RESPONSE TEXT>,
  "tweets": {
    "Document 1": {"index": "4", "score": "1.67938", "text": <MESSAGE TEXT>},
    "Document 2": {"index": "1", "score": "1.611747", "text": <MESSAGE TEXT>},
    "Document 3": {"index": "9", "score": "1.053806", "text": <MESSAGE TEXT>},
    "Document 4": {"index": "13", "score": "1.04539", "text": <MESSAGE TEXT>},
    "Document 5": {"index": "7", "score": "1.017055", "text": <MESSAGE TEXT>}}
}]
```

Figure 87: Sample Genra-RAG response in question answering process.

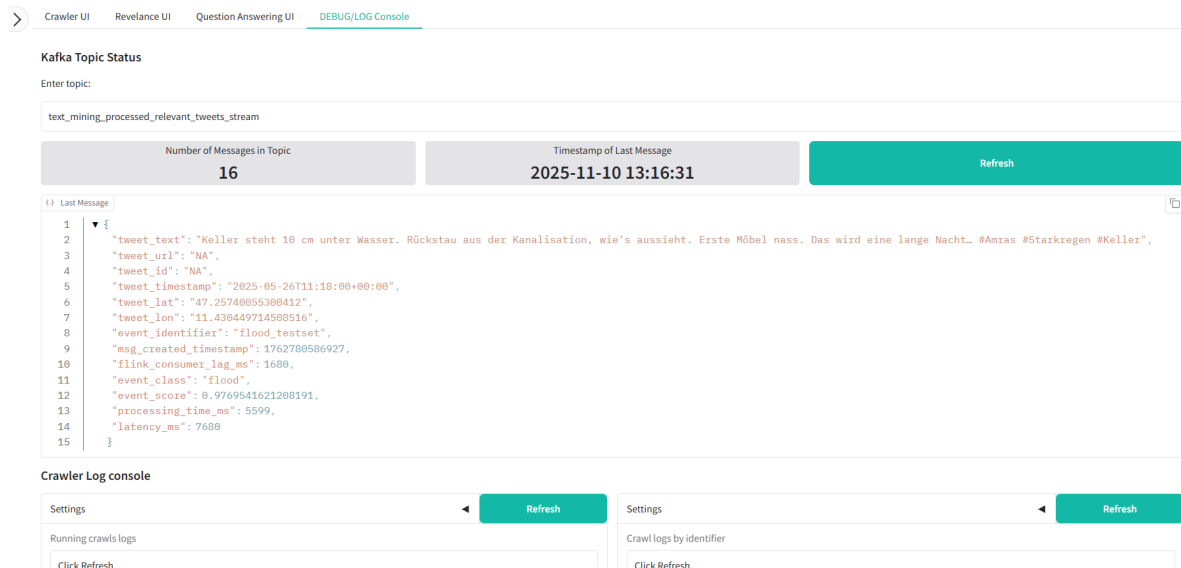


Figure 88: Text mining GUI: Debug and Log console tab

6.6.5 DEBUG/LOG Console Tab

The final tab in the UI is the Debug and Console Logs tab. This tab is primarily used for monitoring messages across different Kafka topics and for tracking crawler logs. It consists of two main sections:

- **Kafka Topic Status:** This section allows users to monitor the contents of a Kafka topic. After entering the topic name in the provided input field and clicking *Refresh*, the console displays the number of messages in the topic, the timestamp of the most recent message, and the contents of that message. This helps users track the topics being populated by the crawler and the event-type prediction components.
- **Crawler Log console:** This section is dedicated to monitoring the crawler logs topic. It displays active crawl instances and the logs generated by the Python script that visits the Twitter/X webpage for data collection.

Figure 88 shows the layout and contents of this tab. Additional implementation details can be found in the text mining GitHub repository.

6.7 Lessons Learned

Mining texts from social media networks and extracting meaningful information is a challenging task—especially when dealing with multilingual posts published during a crisis. First, data accessibility is a major limitation, as direct and complete access to social media content is rarely free. In our social media toolkit, we built a crawler for Twitter/X that collects posts directly from the webpage. However, this approach is inherently limited because it can only access posts in the order they appear on the webpage and cannot guarantee complete retrieval of all relevant content. From this, we learned that for a production-level deployment, it is preferable to use the official Twitter/X API, despite the associated costs, to ensure reliability

and completeness.

During the development of our event type prediction model, we learned that even though social media text is highly unstructured, it is still possible to identify posts containing useful information about weather-related crisis events—provided that the messages contain sufficient textual content to analyze. We also learned that the model may miss certain relevant posts; therefore, although prioritizing posts flagged as relevant is safe, all posts should be reviewed when high recall is essential. Additionally, we found that state-of-the-art LLMs can effectively annotate training data, and their annotations can be further improved through careful label-cleaning procedures. In cases where data is limited for certain languages, LLMs can also generate realistic synthetic posts that emulate messages shared during crisis situations. Finally, we learned that our trained models can be distilled to significantly improve inference efficiency with only minor performance trade-offs.

Furthermore, we combined LLMs with rank aggregation to enhance the effectiveness of zero-shot question answering. This novel QA model outperforms existing zero-shot approaches and in some cases achieving considerable improvements. We observed, however, that these performance gains come at the cost of increased computational demands, particularly as the size of the LLM grows. Nevertheless, the modular nature of the QA framework facilitates further experimentation with different, smaller, possibly better, LLMs and rank aggregation methods.

The social media toolkit described in this section should be used primarily as a complement to existing information sources and not as the sole basis for making operational or actionable decisions during a weather-related crisis.

7 Conclusions

According to the results presented in Section 2, we can state that we have fully achieved the objectives of task 4.1 (see Table 20). We can successfully forecast critical events from multiple sensors with an accuracy above 75% at least 30 minutes ahead of time. Even in the toughest cases, we can predict multiple overflows 30 minutes ahead of time with an accuracy of 80%.

Table 20: KPIs for task 4.1.

[R5]: Multiresolution Complex Event Forecasting Component	SC: 3, V: $\approx 7\text{M } \text{€}$	Forecasting >5 critical business events 60 to 15min ahead of time with 75 to 95 % accuracy
---	---	--

We have additionally presented work on Complex Event Forecasting towards other various directions, besides simply achieving the project's objectives. For the first half of the project (see also D4.1), we focused on adding more expressive power to our forecasting engine and making it adaptive to drifts. For the second half of the project, we focused on two directions:

1. We developed an event forecasting method based on forward recognition. This is the method that allowed us to achieve our objectives.
2. We developed a framework for hierarchical complex event recognition as a first step to an alternative approach of multi-resolution forecasting.

This underlines the fact that Complex Event Forecasting is still a relatively unexplored area, despite the fact that time series forecasting is well-established. Complex Event Forecasting can borrow several ideas from time series forecasting, but these ideas cannot be simply copied. There is a need for extensive experimentation, both theoretical and practical.

Regarding the development of interactive learning approaches for guiding large-scale simulations exploration, and based on the requirements analysis for each use-case performed during the first half of the project (D4.1), we implemented and tested an arsenal of algorithms. In particular, we tested techniques for the calibration of simulators and also for the effective search of relevant parameter spaces. Results from trials indicate that (a) interactive learning is a very practical approach for simulation exploration, given that high-resolution data relevant for each simulator are in place (regarding e.g. weather events, topology maps of interesting areas, assessments on health emergencies, etc.) in order for the simulations to be as realistic as possible; (b) AI and ML algorithms manage to optimize time and performance regarding the tuning of parameters that should be examined in each case, but can also be used in scenarios that are not identical to the use-cases examined in CREXDATA; (c) reinforcement learning in particular can be proven very valuable for systematically exploring intervention strategies, though requiring thorough training to reach stable results.

Regarding the development of novel, communication efficient techniques for federated learning, we can clearly state that we have not only achieved the objectives and KPIs of Task 4.3, but also made significantly more contributions than mentioned at the Description of Work. In

particular:

1. The KPI for achieving a 10x communication reduction was already achieved at Deliverable D4.1, where we displayed communication reductions greatly surpassing 1 order of magnitude, and often reaching 2-3 orders of magnitude.
2. Similarly, the KPI for achieving a 5-10x improvement in training speed was already achieved by Deliverable D4.1, where we displayed over 1 order of magnitude fewer training steps compared to the state-of-the-art FedOpt algorithm.
3. The KPI/requirement about real-time adaptation to concept-drifts is achieved by the design of our FDA and FDA-OPT algorithms, which only require communication when the global model may have changed significantly.
4. The KPI about extending the network lifetime by 2-10x is achieved if we only consider the energy cost of the computation and communication, since both of them are reduced by over 10 times compared to existing techniques.

Additionally, we have performed work not mentioned at the Description of Work. In particular, the integration with the NS3 simulator and the development of novel, near real-time techniques for NeRF computation were additional work, with our NeRF techniques requiring a very significant work on Computer Vision techniques (which were not within the scope of Task 4.3), on top of our federated learning techniques.

According to the results presented in Section 5.5.6 and Figure 66, we can state that we have fully achieved the objectives of Task 4.4 (see Table 21). On both the real testbed and the simulated setups, under various workflows – some of which included training and inference operators – the APEROL and DAG* algorithms achieve on average 5× end-to-end latency improvement on the output execution plans over existing state-of-the-art approaches, while reaching 10× in certain cases.

Table 21: KPIs for task 4.4.

[R6]: PaaS Optimizer Component	SC: 8, V: $\approx 17\text{M €}$	5-10x comm. cost/latency/energy reduction, training, inference & forecasting speed
--------------------------------	--	--

For the first half of the project (see also D4.1), we focused on parallel optimization algorithm design and we conceptualized a first version of the APEROL suite. For the second half of the project, we focused on two directions:

- We designed algorithms that guarantee the suggestion of an optimal execution plan by considerably pruning otherwise enormous search spaces.
- We exhaustively evaluated all our algorithms. Our evaluation included a total of 5 workflows across 2 well established benchmarks. More than 10 different network setups across simulated and real testbed. Comparisons of APEROL, DAG* against 2 candidate, state-of-the-art approaches. To our knowledge, this is the first time such a comprehensive experimental evaluation has been compiled.

What is more, is that our algorithmic designs have the potential to further affect research and development in this area, by coming up with hybrid approaches [70] or providing novel admissible heuristics [74].

In the case of the social media toolkit, the components detailed in Section 6 fulfill the objectives of Task 4.5, which align with the main objective (MO1.3) of handling multilingual social media data in real time. We are able to crawl posts within seconds of publication and classify relevant messages with an overall system latency of approximately 7 seconds—within range of the latency KPI (see Table 22). Furthermore, we can extract additional value from these posts using our LLM-enabled question answering component.

Table 22: KPIs for task 4.5.

[R3]: Multilingual Social Media Toolkit	SC: 1, V: $\approx 1\text{M €}$	Delivering multilingual social media context to CREXDATA with <5 sec latency from the time of the post for geo-locations of interest
---	---	--

During the first half of the project (see D4.1), we focused on experimentation and methodology development for both the multilingual language model and GENRA, our question answering process. In the second half, we concentrated on integrating the components into a cohesive social media toolkit, improving our models, and developing a text mining UI to interface with the tools.

8 Acronyms and Abbreviations

- CA – Consortium Agreement
- D – deliverable
- DoA – Description of Action (Annex 1 of the Grant Agreement)
- EB – Executive Board
- EC – European Commission
- GA – General Assembly / Grant Agreement
- HPC - High Performance Computing
- UDF - User Defined Function
- WP – Work Package
- WPL – Work Package Leader

References

- [1] Cédric Adjih, Emmanuel Baccelli, Eric Fleury, Gaetan Harter, Nathalie Mitton, Thomas Noel, Roger Pissard-Gibollet, Frédéric Saint-Marcel, Guillaume Schreiner, Julien Vandaele, and Thomas Watteyne. "FIT IoT-LAB: A Large Scale Open Experimental IoT Testbed". Milan, Italy, December 2015.
- [2] Elias Alevizos, Alexander Artikis, and Georgios Paliouras. Complex event forecasting with prediction suffix trees. *VLDB J.*, 31(1):157–180, 2022.
- [3] Elias Alevizos, Alexander Artikis, and Georgios Paliouras. Complex event recognition with symbolic register transducers. *Proc. VLDB Endow.*, 17(11):3165–3177, 2024.
- [4] Ni An, Guanqi Wang, Di Wang, Gang Ma, Xiaolin Chang, and Wei Zhou. Dem parameter calibration based on multi-objective bayesian optimization and prior physical information. *Acta Geotechnica*, 20(3):1379–1401, 2025.
- [5] Alex Arenas, Wesley Cota, Jesús Gómez-Gardeñes, Sergio Gómez, Clara Granell, Joan T. Matamalas, David Soriano-Paños, and Benjamin Steinegger. Modeling the spatiotemporal epidemic spreading of covid-19 and the impact of mobility and social distancing interventions. *Phys. Rev. X*, 10(4):041055, 2020.
- [6] Francesco Barbieri, Luis Espinosa Anke, and Jose Camacho-Collados. XLM-T: Multilingual language models in Twitter for sentiment analysis and beyond. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 258–266, Marseille, France, June 2022. European Language Resources Association.
- [7] Xenofon Chatziliadis, Eleni Tziritza Zacharatou, Alphan Eracar, Steffen Zeuch, and Volker Markl. Efficient placement of decomposable aggregation functions for stream processing over large geo-distributed topologies. *Proc. VLDB Endow.*, 17(6):1501–1514, February 2024.
- [8] Ankit Chaudhary, Steffen Zeuch, and Volker Markl. Governor: Operator placement for a unified fog-cloud environment. In Angela Bonifati, Yongluan Zhou, Marcos Antonio Vaz Salles, Alexander Böhm, Dan Olteanu, George H. L. Fletcher, Arijit Khan, and Bin Yang, editors, *Proceedings of the 23rd International Conference on Extending Database Technology, EDBT 2020, Copenhagen, Denmark, March 30 - April 02, 2020*, pages 631–634. OpenProceedings.org, 2020.
- [9] S. Chintapalli, D. Dagit, B. Evans, R. Farivar, T. Graves, M. Holderbaugh, Z. Liu, K. Nusbbaum, K. Patil, B. J. Peng, et al. Benchmarking Streaming Computation Engines: Storm, Flink and Spark Streaming. In *Proceedings of the IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 1789–1792. IEEE, 2016.
- [10] Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. *CoRR*, abs/1911.02116, 2019.

- [11] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, and Daniel Campos. Overview of the TREC 2020 deep learning track, February 2021. arXiv:2102.07662 [cs].
- [12] Gianpaolo Cugola and Alessandro Margara. Processing flows of information: From data stream to complex event processing. *ACM Comput. Surv.*, 44(3):15:1–15:62, 2012.
- [13] Loris D’Antoni and Margus Veanes. The power of symbolic automata and transducers. In *CAV (1)*, volume 10426 of *Lecture Notes in Computer Science*, pages 47–67. Springer, 2017.
- [14] Peter Dayan and CJCH Watkins. Q-learning. *Machine learning*, 8(3):279–292, 1992.
- [15] Alan J. Demers, Johannes Gehrke, Mingsheng Hong, Mirek Riedewald, and Walker M. White. Towards expressive publish/subscribe systems. In *EDBT*, volume 3896 of *Lecture Notes in Computer Science*, pages 627–644. Springer, 2006.
- [16] Alan J. Demers, Johannes Gehrke, Biswanath Panda, Mirek Riedewald, Varun Sharma, and Walker M. White. Cayuga: A general purpose event monitoring system. In *CIDR*, pages 412–422. www.cidrdb.org, 2007.
- [17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Tamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.
- [18] William B. Dolan and Chris Brockett. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*, 2005.
- [19] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for on-line learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011.
- [20] European Union-Copernicus Marine Service. Global ocean 1/12° physics analysis and forecast updated daily, 2024.
- [21] European Union-Copernicus Marine Service. Global ocean waves analysis and forecast, 2024.
- [22] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1126–1135. PMLR, 06–11 Aug 2017.
- [23] National Centers for Environmental Prediction (NCEP). Noaa global forecast system (gfs) wind data, 2024.
- [24] Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. Precise Zero-Shot Dense Retrieval without Relevance Labels, December 2022. arXiv:2212.10496 [cs].

- [25] Nadia Mushtaq Gardazi, Ali Daud, Muhammad Kamran Malik, Amal Bukhari, Tariq Alsahfi, and Bader Alshemaimri. BERT applications in natural language processing: a review. *Artif. Intell. Rev.*, 58(6), March 2025.
- [26] Nikos Giatrakos, Elias Alevizos, Alexander Artikis, Antonios Deligiannakis, and Minos N. Garofalakis. Complex event recognition in the big data era: a survey. *VLDB J.*, 29(1):313–352, 2020.
- [27] Georgios Grigoropoulos, Giannis Spiliopoulos, Ilias Chamatidis, Manolis Kaliorakis, Alexandros Troupiotis-Kapeliaris, Marios Vodas, Evangelia Filippou, Eva Chondrodima, Nikos Pelekis, Yannis Theodoridis, Dimitris Zissis, and Konstantina Bereta. A scalable system for maritime route and event forecasting. In *Proceedings of the 27th International Conference on Extending Database Technology (EDBT)*, New York, NY, USA, 2024. Association for Computing Machinery.
- [28] Harshit Gupta, Amir Vahid Dastjerdi, Soumya K. Ghosh, and Rajkumar Buyya. ifogsim: A toolkit for modeling and simulation of resource management techniques in the internet of things, edge and fog computing environments. *Software: Practice and Experience*, 47(9):1275–1296, 2017.
- [29] Pengcheng He, Jianfeng Gao, and Weizhu Chen. DeBERTaV3: Improving DeBERTa using electra-style pre-training with gradient-disentangled embedding sharing, 2023.
- [30] Tzu-Ming Harry Hsu, Hang Qi, and Matthew Brown. Measuring the effects of non-identical data distribution for federated visual classification. *CoRR*, 2019.
- [31] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.
- [32] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D’Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badi Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Hang Qi, Daniel Ramage, Ramesh Raskar, Mariana Raykova, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, 14(1–2):1–210, 6 2021.
- [33] Rohit Kumar Kaliyar. A multi-layer bidirectional transformer encoder for pre-trained word embedding: A survey of bert. In *2020 10th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, pages 336–340, 2020.
- [34] Zoi Kaoudi, Jorge-Arnulfo Quiané-Ruiz, Bertty Contreras-Rojas, Rodrigo Pardo-Meza, Anis Troudi, and Sanjay Chawla. ML-based cross-platform query optimization. In *36th*

- IEEE International Conference on Data Engineering, ICDE 2020, Dallas, TX, USA, April 20-24, 2020*, pages 1489–1500. IEEE, 2020.
- [35] Jeyhun Karimov, Tilmann Rabl, Asterios Katsifodimos, Roman Samarev, Henri Heiskanen, and Volker Markl. Benchmarking Distributed Stream Data Processing Systems. In *34th IEEE International Conference on Data Engineering, ICDE 2018, Paris, France, April 16-19, 2018*, pages 1507–1518. IEEE Computer Society, 2018.
- [36] Georgios Katsimpras and Georgios Paliouras. Genra: Enhancing zero-shot retrieval with rank aggregation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 7566–7577, 2024.
- [37] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, San Diego, CA, USA, 2015.
- [38] Andreas Kouvaras, Periklis Mantenoglou, and Alexander Artikis. Generating activity definitions with large language models. 2025.
- [39] Huizhen Kuang and Lingli Wang. Multi-objective design space exploration for high-level synthesis via bayesian optimization. In *2023 International Symposium of Electronics Design Automation (ISED)*, pages 150–155. IEEE, 2023.
- [40] Qinbin Li, Yiqun Diao, Quan Chen, and Bingsheng He. Federated learning on non-iid data silos: An experimental study. In *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*, pages 965–978. IEEE, 2022.
- [41] Bryan Lim, Sercan Ö Arık, Nicolas Loeff, and Tomas Pfister. Temporal fusion transformers for interpretable multi-horizon time series forecasting. *International journal of forecasting*, 37(4):1748–1764, 2021.
- [42] Bill Yuchen Lin, Chaoyang He, Zihang Ze, Hulin Wang, Yufen Hua, Christophe Dupuy, Rahul Gupta, Mahdi Soltanolkotabi, Xiang Ren, and Salman Avestimehr. FedNLP: Benchmarking federated learning methods for natural language processing tasks. In Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz, editors, *Findings of the Association for Computational Linguistics: NAACL 2022*, pages 157–175, Seattle, United States, July 2022. Association for Computational Linguistics.
- [43] Mo Liu, Elke A. Rundensteiner, Kara Greenfield, Chetan Gupta, Song Wang, Ismail Ari, and Abhay Mehta. E-cube: multi-dimensional event sequence analysis using hierarchical pattern query sharing. In *SIGMOD Conference*, pages 889–900. ACM, 2011.
- [44] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692, 2019.
- [45] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.
- [46] Md. Redowan Mahmud, Samodha Pallewatta, Mohammad Goudarzi, and Rajkumar Buyya. ifogsim2: An extended ifogsim simulator for mobility, clustering, and microservice

- management in edge and fog computing environments. *J. Syst. Softw.*, 190:111351, 2022.
- [47] Periklis Mantenoglou and Alexander Artikis. Composite event recognition with arbitrary specifications. *Information and Computation*, page 105373, 2025.
- [48] Periklis Mantenoglou and Alexander Artikis. Sequencing in the run-time event calculus. *European Conference on Artificial Intelligence (ECAI)*, 2025.
- [49] Periklis Mantenoglou and Alexander Artikis. Temporal specification optimisation for the event calculus. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 15075–15082, 2025.
- [50] Periklis Mantenoglou, Dimitrios Kelesis, and Alexander Artikis. Complex event recognition with allen relations. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, volume 19, pages 502–511, 2023.
- [51] Periklis Mantenoglou, Manolis Pitsikalis, and Alexander Artikis. Reasoning over streams of events with delayed effects. *Journal of Artificial Intelligence Research*, 84, 2025.
- [52] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In Aarti Singh and Jerry Zhu, editors, *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, volume 54 of *Proceedings of Machine Learning Research*, pages 1273–1282. PMLR, 4 2017.
- [53] Yuan Mei and Samuel Madden. Zstream: a cost-based query processor for adaptively detecting composite events. In *SIGMOD Conference*, pages 193–206. ACM, 2009.
- [54] Charilaos Akasiadis Miguel Ponce de León, Isabel Martínez. Epi-sim/episim-rl: epidemic simulator coupled with reinforcement learning techniques, 2025. Release-crexdata.
- [55] Pedro Velho Miguel Ponce-de León, Isabel Martínez. Episim-emews: epidemic simulator exploration workflow, 2025. Version 1.0.0.
- [56] Thomas Müller, Alex Evans, Christoph Schied, and Alexander Keller. Instant neural graphics primitives with a multiresolution hash encoding. *ACM Transactions on Graphics*, 41(4):1–15, July 2022.
- [57] Mihai Nadăș, Laura Dioșan, and Andreea Tomescu. Synthetic data generation using large language models: Advances in text and code. *IEEE Access*, 13:134615–134633, 2025.
- [58] Curtis Northcutt, Lu Jiang, and Isaac Chuang. Confident learning: Estimating uncertainty in dataset labels. 70:1373–1411, April 2021.
- [59] Jonathan Ayebakuro Orama, Marc Juvillà Garcia, and Maite Melero. Evaluating instruction-tuned large language models for automatic annotation of multilingual climate disaster-related social media messages. In *Frontiers in Artificial Intelligence and Applications*, Frontiers in artificial intelligence and applications. IOS Press, September 2025.
- [60] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison,

- Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. *PyTorch: an imperative style, high-performance deep learning library*. Curran Associates Inc., Red Hook, NY, USA, 2019.
- [61] Ronak Pradeep, Sahel Sharifymoghaddam, and Jimmy Lin. Rankvicuna: Zero-shot listwise document reranking with open-source large language models. *arXiv preprint arXiv:2309.15088*, 2023.
- [62] Zhen Qin, Shuiguang Deng, Mingyu Zhao, and Xueqiang Yan. Fedapen: Personalized cross-silo federated learning with adaptability to statistical heterogeneity. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining, KDD '23*, page 1954–1964, New York, NY, USA, 2023. Association for Computing Machinery.
- [63] Sashank J. Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečný, Sanjiv Kumar, and Hugh Brendan McMahan. Adaptive federated optimization. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*, 2021.
- [64] Lydia Wahid Rizkallah. Optimizing svm hyperparameters for satellite imagery classification using metaheuristic and statistical techniques. *International Journal of Data Science and Analytics*, pages 1–18, 2025.
- [65] Melrose Roderick, James MacGlashan, and Stefanie Tellex. Implementing the deep q-network. *arXiv preprint arXiv:1711.07478*, 2017.
- [66] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108, 2019.
- [67] Patrick Schäfer and Ulf Leser. Multivariate time series classification with weasel+ muse. *arXiv preprint arXiv:1711.11343*, 2017.
- [68] Patrick Schäfer and Ulf Leser. Teaser: early and accurate time series classification. *Data mining and knowledge discovery*, 34(5):1336–1362, 2020.
- [69] Anshu Shukla, Shilpa Chaturvedi, and Yogesh Simmhan. Riotbench: An iot benchmark for distributed stream processing systems. *Concurrency and Computation: Practice and Experience*, 29(21):e4257, 2017.
- [70] D Banelas A Simitsis and N Giatrakos. Aperol: Adaptive parallel edge-to-cloud runtime optimization for layered workflow execution. *VLDB*, 2026.
- [71] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In David Yarowsky, Timothy Baldwin, Anna Korhonen, Karen Livescu, and Steven Bethard, editors, *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.
- [72] Giannis Spiliopoulos, Marios Vodas, Georgios Grigoropoulos, Konstantina Bereta, and Dimitris Zissis. Patterns of life: Global inventory for maritime mobility patterns. In *Proceedings of the 27th International Conference on Extending Database Technol-*

- ogy (*EDBT*), pages 770–777, New York, NY, USA, 2024. Association for Computing Machinery.
- [73] Vasileios Stavropoulos, Evangelos Michelioudakis, Charilaos Akasiadis, and Alexander Artikis. Resource-effective exploration of tumor treatments with multi-scale simulations. In *Proceedings of the 12th Hellenic Conference on Artificial Intelligence*, pages 1–10, 2022.
- [74] Errikos Streviniotis, Dimitrios Banelas, Nikos Giatrakos, and Antonios Deligiannakis. DAG*: A novel a*-like algorithm for optimal workflow execution across iot platforms. In *Proceedings of the 41st International Conference on Data Engineering (ICDE’25)*, Hong Kong SAR, China, May 2025.
- [75] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, 6 2013. PMLR.
- [76] Uber Technologies. Uber h3: Uber’s hexagonal hierarchical spatial index, 2019.
- [77] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. BEIR: A Heterogenous Benchmark for Zero-shot Evaluation of Information Retrieval Models, October 2021. arXiv:2104.08663 [cs].
- [78] Michail Theologitis, Georgios Frangias, Georgios Anestis, Vasilis Samoladas, and Antonios Deligiannakis. Communication-efficient distributed deep learning via federated dynamic averaging. In *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain, March 25 - March 28*, page 411–424. Open-Proceedings.org, 2025.
- [79] Michail Theologitis, Vasilis Samoladas, and Antonios Deligiannakis. Communication-efficient federated fine-tuning of language models via dynamic update schedules, 2025.
- [80] Haithem Turki, Deva Ramanan, and Mahadev Satyanarayanan. Mega-nerf: Scalable construction of large-scale nerfs for virtual fly-throughs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 12922–12931, June 2022.
- [81] Marianna Tzortzi, Charalampos Kleitsikas, Agis Politis, Sotirios Niarchos, Katerina Doka, and Nectarios Koziris. Planning workflow executions over the edge-to-cloud continuum. In Ioannis Chatzigiannakis and Ioannis Karydis, editors, *Algorithmic Aspects of Cloud Computing - 8th International Symposium, ALGO CLOUD 2023, Amsterdam, The Netherlands, September 5, 2023, Revised Selected Papers*, volume 14053 of *Lecture Notes in Computer Science*, pages 9–24. Springer, 2023.
- [82] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

- [83] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In Tal Linzen, Grzegorz Chrupała, and Afra Alishahi, editors, *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355, Brussels, Belgium, November 2018. Association for Computational Linguistics.
- [84] Jianyu Wang, Zachary Charles, Zheng Xu, Gauri Joshi, H. Brendan McMahan, Blaise Agüera y Arcas, Maruan Al-Shedivat, Galen Andrew, Salman Avestimehr, Katharine Daly, Deepesh Data, Suhas N. Diggavi, Hubert Eichner, Advait Gadhikar, Zachary Garrett, Antonious M. Girgis, Filip Hanzely, Andrew Hard, Chaoyang He, Samuel Horváth, Zhouyuan Huo, Alex Ingerman, Martin Jaggi, Tara Javidi, Peter Kairouz, Satyen Kale, Sai Praneeth Karimireddy, Jakub Konečný, Sanmi Koyejo, Tian Li, Luyang Liu, Mehryar Mohri, Hang Qi, Sashank J. Reddi, Peter Richtárik, Karan Singhal, Virginia Smith, Mahdi Soltanolkotabi, Weikang Song, Ananda Theertha Suresh, Sebastian U. Stich, Ameet Talwalkar, Hongyi Wang, Blake E. Woodworth, Shanshan Wu, Felix X. Yu, Honglin Yuan, Manzil Zaheer, Mi Zhang, Tong Zhang, Chunxiang Zheng, Chen Zhu, and Wennan Zhu. A field guide to federated optimization. *CoRR*, abs/2107.06917, 2021.
- [85] Zhiqiang Wang, Yiran Pang, and Yanbin Lin. Large language models are zero-shot text classifiers. 2023.
- [86] Walker M. White, Mirek Riedewald, Johannes Gehrke, and Alan J. Demers. What is "next" in event processing? In *PODS*, pages 263–272. ACM, 2007.
- [87] Adina Williams, Nikita Nangia, and Samuel Bowman. A broad-coverage challenge corpus for sentence understanding through inference. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 1112–1122. Association for Computational Linguistics, 2018.
- [88] Herbert Woisetschlager, Alexander Erben, Shiqiang Wang, Ruben Mayer, and Hans-Arno Jacobsen. Federated fine-tuning of llms on the very edge: The good, the bad, the ugly. In *Proceedings of the Eighth Workshop on Data Management for End-to-End Machine Learning, DEEM '24*, page 39–50, New York, NY, USA, 2024. Association for Computing Machinery.
- [89] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-art natural language processing. In Qun Liu and David Schlangen, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics.
- [90] Seungyoon Woo, Junhyeog Yun, and Gunhee Kim. Meta-continual learning of neural fields. In Y. Yue, A. Garg, N. Peng, F. Sha, and R. Yu, editors, *International Conference on Representation Learning*, volume 2025, pages 56076–56098, 2025.

- [91] Xin Zhang, Yanzhao Zhang, Dingkun Long, Wen Xie, Ziqi Dai, Jialong Tang, Huan Lin, Baosong Yang, Pengjun Xie, Fei Huang, et al. mgte: Generalized long-context text representation and reranking models for multilingual text retrieval. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 1393–1412, 2024.
- [92] Xinyang Zhang, Yury Malkov, Omar Florez, Serim Park, Brian McWilliams, Jiawei Han, and Ahmed El-Kishky. Twbin-bert: A socially-enriched pre-trained language model for multilingual tweet representations. *arXiv preprint arXiv:2209.07562*, 2022.
- [93] Zhuo Zhang, Yuanhang Yang, Yong Dai, Qifan Wang, Yue Yu, Lizhen Qu, and Zenglin Xu. FedPETuning: When federated learning meets the parameter-efficient tuning methods of pre-trained language models. In Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki, editors, *Findings of the Association for Computational Linguistics: ACL 2023*, pages 9963–9977, Toronto, Canada, July 2023. Association for Computational Linguistics.
- [94] Ce Zhou, Qian Li, Chen Li, Jun Yu, Yixin Liu, Guangjing Wang, Kai Zhang, Cheng Ji, Qiben Yan, Lifang He, Hao Peng, Jianxin Li, Jia Wu, Ziwei Liu, Pengtao Xie, Caiming Xiong, Jian Pei, Philip S Yu, and Lichao Sun. A comprehensive survey on pretrained foundation models: a history from BERT to ChatGPT. *Int. J. Mach. Learn. Cybern.*, November 2024.